AFRL-AFOSR-VA-TR-2016-0023

PARADIGMS FOR EMERGENCE OF SHAPE AND FUNCTION IN BIOMOLECULAR ELEXTROLYTES FOR THE DESIGN OF BIOMIMETIC MATERIALS

Monica Olvera De La Cruz
NORTHWESTERN UNIVERSITY

12/04/2015
Final Report

Air Force Research Laboratory
AF Office Of Scientific Research (AFOSR)/ RTA1
Arlington, Virginia 22203
Air Force Materiel Command

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
11/18/2015	Final Performance Report	April 2010 - August 2015

4. TITLE AND SUBTITLE

Paradigms for Emergence of Shape and Function in Biomolecular Electrolytes for the Design of Biomimetic Materials.

5a. CONTRACT NUMBER

5b. GRANT NUMBER

FA9550-10-1-0167

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Olvera de la Cruz, Monica

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Northwestern University
2220 Campus Drive
Evanston, IL 60208

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

DISTRIBUTION A

13. SUPPLEMENTARY NOTES

14. ABSTRACT

We discovered the buckling of elastically heterogeneous nano-containers into regular and irregular polyhedral geometries observed in organelles of multicomponent proteins and in halophilic organisms. We determined the physical properties of ionic membranes. By co-assembling cationic and anionic amphiphiles programmed to form ionic crystalline vesicles we demonstrated the buckling model and produced biomimetic organelles. We extended the model to describe the blebbing of nuclear lamina in cells in patients with cancer and other serious pathologies, and discovered that blebbing is due to the segregation of two main lamins types that form the elastic membranes that protect the nucleus of cells. We discovered long-range attractions mediated by concentrated electrolytes. These depletion-like interactions are responsible for the well known "salting out" phenomena. We provided the rules for efficient self-replication of colloidal dimers. We demonstrated that the input of cyclic energy leads to maximum exponential growth. We discovered the functional required to performed molecular dynamics studies of charged components in dielectrically heterogeneous media, and developed efficient codes to analyze concentrated electrolyte solutions. We trained a large number of students and

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(Include area code)*

Reset

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

INSTRUCTIONS FOR COMPLETING SF 298

1. REPORT DATE. Full publication date, including day, month, if available. Must cite at least the year and be Year 2000 compliant, e.g. 30-06-1998; xx-06-1998; xx-xx-1998.

2. REPORT TYPE. State the type of report, such as final, technical, interim, memorandum, master's thesis, progress, quarterly, research, special, group study, etc.

3. DATES COVERED. Indicate the time during which the work was performed and the report was written, e.g., Jun 1997 - Jun 1998; 1-10 Jun 1996; May - Nov 1998; Nov 1998.

4. TITLE. Enter title and subtitle with volume number and part number, if applicable. On classified documents, enter the title classification in parentheses.

5a. CONTRACT NUMBER. Enter all contract numbers as they appear in the report, e.g. F33615-86-C-5169.

5b. GRANT NUMBER. Enter all grant numbers as they appear in the report, e.g. AFOSR-82-1234.

5c. PROGRAM ELEMENT NUMBER. Enter all program element numbers as they appear in the report, e.g. 61101A.

5d. PROJECT NUMBER. Enter all project numbers as they appear in the report, e.g. 1F665702D1257; ILIR.

5e. TASK NUMBER. Enter all task numbers as they appear in the report, e.g. 05; RF0330201; T4112.

5f. WORK UNIT NUMBER. Enter all work unit numbers as they appear in the report, e.g. 001; AFAPL30480105.

6. AUTHOR(S). Enter name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. The form of entry is the last name, first name, middle initial, and additional qualifiers separated by commas, e.g. Smith, Richard, J, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES). Self-explanatory.

8. PERFORMING ORGANIZATION REPORT NUMBER. Enter all unique alphanumeric report numbers assigned by the performing organization, e.g. BRL-1234; AFWL-TR-85-4017-Vol-21-PT-2.

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES). Enter the name and address of the organization(s) financially responsible for and monitoring the work.

10. SPONSOR/MONITOR'S ACRONYM(S). Enter, if available, e.g. BRL, ARDEC, NADC.

11. SPONSOR/MONITOR'S REPORT NUMBER(S). Enter report number as assigned by the sponsoring/ monitoring agency, if available, e.g. BRL-TR-829; -215.

12. DISTRIBUTION/AVAILABILITY STATEMENT. Use agency-mandated availability statements to indicate the public availability or distribution limitations of the report. If additional limitations/ restrictions or special markings are indicated, follow agency authorization procedures, e.g. RD/FRD, PROPIN, ITAR, etc. Include copyright information.

13. SUPPLEMENTARY NOTES. Enter information not included elsewhere such as: prepared in cooperation with; translation of; report supersedes; old edition number, etc.

14. ABSTRACT. A brief (approximately 200 words) factual summary of the most significant information.

15. SUBJECT TERMS. Key words or phrases identifying major concepts in the report.

16. SECURITY CLASSIFICATION. Enter security classification in accordance with security classification regulations, e.g. U, C, S, etc. If this form contains classified information, stamp classification level on the top and bottom of this page.

17. LIMITATION OF ABSTRACT. This block must be completed to assign a distribution limitation to the abstract. Enter UU (Unclassified Unlimited) or SAR (Same as Report). An entry in this block is necessary if the abstract is to be limited.

AFOSR Final Performance Report

Project Title: Paradigms for Emergence of Shape and Function in Biomolecular Electrolytes for the Design of Biomimetic Materials.

Award Number: FA9550-10-1-0167

Start Date: May 1, 2010

Program Manager: Dr. Julie Moses

Air Force Office of Scientific Research
875 North Randolph Street
Suite 325, Room 3112
Arlington, Virginia 22203-1768

Email: Julie.Moses@afosr.af.mil
Phone: (703) 696-9586

Principal Investigator: Prof. Monica Olvera de la Cruz
Dept. of Materials Science and Engineering
Northwestern University
Evanston, IL 60208

Email: m-olvera@northwestern.edu
Phone: (847) 491-7801

Accomplishments/New Findings:

- We report a GPU-implementation in HOOMD Blue of long-range electrostatic interactions based on the orientation-averaged Ewald sum scheme. Our GPU implementation is significantly faster than the CPU implementation of the Ewald method for small to a sizable number of particles ($\sim 10^5$) and underlies all our fundamental numerical and computational research in the proposal.
- We formulated a generalized elastic model for inhomogeneous shells, we demonstrate that co-assembled shells with two elastic components buckle into polyhedra such as dodecahedra, octahedra, tetrahedra, and hosohedra shells via a mechanism that explains many observations, predicts a new family of polyhedral shells, and provides the principles for designing micro-containers with specific shapes and symmetries for numerous applications in materials and life sciences. We have used simulated annealing Monte Carlo simulations to determine optimal shapes and domain distributions of a two-component elastic shell over a wide range of inter-component line tensions, relative concentrations of the components, as well as relative bending rigidities. We found a rich gallery of shapes, including a number of polyhedral structures at moderate values of the line tension and the bending rigidity ratio. Our analysis shows that it is in principle possible to combine two materials with different elastic properties to construct a wide variety of shell shapes with potential applications in medical imaging, drug delivery, or composite material design. Our model assumes the presence of the 12 disclination defects originating from the construction of a crystal lattice on a sphere. However, our results suggest that the faceting into polyhedral shapes is not directly related to the defects, even though their presence introduces strain to the shell.
- We showed that an explicit solution of the Poisson equation can be entirely avoided in simulations that originate from a judiciously designed variational formulation;
- We showed that an explicit solution of the Poisson equation can be entirely avoided in simulations that originate from a judiciously designed variational formulation;
  We have developed a variational formulation of electrostatics that uses the ideal combination of the functional and the optimization scheme to allow for large-scale simulation of charges in the presence of varying dielectric response. The functional that we derive is a true energy functional, yielding the correct thermodynamic force during the approach to the minimum and making the full exploitation of dynamical optimization procedures possible. Apart from the standard assumption of linear dielectric response, we make no assumptions about the electrostatic system in the derivation of the functional. This makes possible the more ambitious long-term goal of investigating the effects of evolving geometry, which underlies a deeper study of the interplay between geometry and electrostatic interactions.
- We have characterized the mechanical properties of multi-component fibers. The specific fibers under study consist of two components with different bending and twist stiffness, length, and spontaneous curvature. We have developed a theoretical formalism for predicting the conformation of these fibers in an enormous space of mechanical properties. Ultimately, our work develops a catalog of fiber conformations as a function of a variety of mechanical input conditions.
- We have investigated and provided multiple support arguments regarding the hypothesis that polyvalent oligonucleotide-functionalized gold nanoparticles are remarkably stable in a cellular environment against degradation by nucleases due to the local high concentration of mono- and divalent ions. In particular, we investigate the composition of the ion cloud around spherical nanoparticles that are functionalized by stiff, highly charged polyelectrolyte

chains by means of classical density functional theory and molecular dynamics simulations, and present a cell model that includes ligands explicitly and is both applicable over the entire relevant parameter space and in excellent quantitative agreement with simulations.

- We reported a model that demonstrates the separation (by reverse osmosis, capillary electrophoresis, or chromatography) of cationic nanoparticles, including proteins with equal net charge but different surface charge distributions.
- We showed that charge correlations between the head groups can lead to solidification of a lipid bilayer that would be in a liquid state if the charges were absent. All elastic parameters of the bilayer such as the bending rigidity κ and the two-dimensional Young's modulus Y are found to depend on the relative charge stoichiometric ratio between the lipid head groups.
- We used an elastic model to explore faceting of solid-wall vesicles with elastic heterogeneities, and showed that faceting occurs in regions where the vesicle wall is softer, such as areas of reduced wall thicknesses or concentrated in crystalline defects.
- We studied the force between two identical charged nanoparticles in the presence of an electrolyte with different ionic size theoretically and via molecular simulations. An asymmetric behavior with respect to the colloidal charge polarity was found for the effective interactions between the nanoparticles. In particular, short-range attractions are observed between two equally and highly charged nanoparticles. This unexpected behavior could be relevant for colloidal stability applications, considering that our model does not include specific attractive interactions. These attractions are greatly enhanced for anionic nanoparticles immersed in standard electrolytes where cations are smaller than anions.
- We presented a closure for the Ornstein-Zernike equation, applicable for fluids of charged, hard spheres. From an exact, but intractable closure, we derive the radial distribution function of nonlinear Debye-Hückel theory by subsequent approximations, and use the information to formulate a new closure by an extension of the mean spherical approximation. The radial distribution functions of the new closure, coined Debye-Hückel-extended mean spherical approximation, are in excellent agreement with those resulting from the hyper-netted chain approximation and molecular dynamics simulations, in the regime where the latter are applicable, except for moderately dilute systems at low temperatures where the structure agrees at most qualitatively.
- We have extended the coarse grained model to faithfully mimic relative design parameters in dsDNA systems. Using Molecular Dynamics simulations with nanoparticles in the size range from 8 nm to 15 nm, overall DNA-nanoparticle hydrodynamic radii of 10 nm to 30 nm, and the number of DNA strands per particle between 40 and 100, the model proves robust, such that completely random initial configurations can self-assemble into a variety of crystal structures, including BCC, CsCl, AlB_2 and Cr_3Si. With these simulation data, we have constructed detailed phase diagrams that closely correspond to the experimental results from wet-laboratory studies.
- We described several distinct ion-induced forces in aqueous solutions of monovalent ions at salt concentrations ranging roughly between 0.01 and 1 M that are expected to compete for dominance with the well-known screened Coulomb forces.
- We introduced the concept of soft structure to connect the induced forces to the local microscopic structure in the ionic fluid and visualize the deformation of the local environment around the ions near several types of boundaries that are neutral, adsorbing, charged, polarizable, or a combination of these properties. The considered forces cannot be

found by well-known (practical) mean-field theories and are relevant for neutral solutes or solutes close to an isoelectric point as well.

- We provided an intuitively appealing overview to aid and enhance predictability and control in experiments. The results from the state of the art liquid theory techniques (namely AHNC) may provide guidance for the development of simple scaling laws, potentially efficient and practical Density Functional Theories (DFTs), and coarse-grained simulation methods to overcome the disparate length, time, and energy scales that characterize nanomaterials and biological matter.
- We developed a minimal continuum elastic model of a lamin meshwork that we used to investigate which aspects of the meshwork could be responsible for bleb formation in the cell's nucleus.
- We studied a strongly adsorbed flexible polyelectrolyte chain on tori. The patterns of the adsorbed chain are analyzed in the space of the toroidal coordinates and in terms of the orientation of each chain segment. Various patterns are found, including double spirals, disclination-like structures, Janus tori, and uniform wrappings, arising from the long-range electrostatic interaction and the toroidal geometry.
- We performed atomistic simulations and revealed that the competition of physical interactions and charge-regulation induced transitions in crystalline membrane states that translate in changes of shapes. We demonstrated that the bilayer thickness and molecular packing are not homogeneous in the low symmetry closed shapes, and that the curved regions are less ordered and therefore would allow higher transport of molecules than the crystalline flat polyhedra faces.
- We calculated the electrostatic potential difference, excess surface tension, and differential capacity corresponding to the distribution of ions near liquid interfaces via Monte Carlo simulations. This includes ion correlations and polarization effects for equally sized ions in water, and equally sized ions in oil.
- We studied a dense two-dimensional binary mixture consisting of ionized amphiphiles with positive and negative species. We analyze this co-assembled system by numerically exact optimization and by continuum Monte Carlo simulations including short range and electrostatic interactions among all the particles. We found the optimal structure to be a triangular lattice for high salt concentrations, a face-centered rectangular lattice for intermediate salt concentrations, and a square lattice for low salt concentrations.
- We investigated the effects of topological defects on the low-energy shapes of single-component two-dimensional elastic vesicles with spherical topology. We found that since the volume constraint partially suppresses the buckling transition such that the buckled icosahedral shape has a reduced asphericity, defect scars are favorable over a larger range of elastic parameters of the membrane compared with systems having no constraint on volume.
- We studied the shapes of pored membranes within the framework of the Helfrich theory under the constraints of fixed area and pore size. We showed that the mean curvature term leads to a budding-like structure, while the Gaussian curvature term tends to flatten the membrane near the pore; this is corroborated by simulation.
- We determine how the dielectric heterogeneities of a spherical nanoparticle influence the ion distribution of their surrounding medium.
- We demonstrate that electrolyte concentration can be used as a tool to manipulate the shape of soft, closed membranes and provide exact expressions for the electrostatic energy of uniformly charged prolate and oblate spheroidal shells.

- We extended the electrostatic analysis from charged vesicles to charged ribbons which were recently synthesized in experiments with promising applications in realizing several biological functions and designing new structures in nanotechnologies. Simulations showed the hierarchical buckling of the ribbon from its initially flat shape to the undulated and then to the out-of-plane twisted conformations with the variation of screening length. Using the tool of classical differential geometry, we found the geometric origin of the electrostatically driven conformal transformation. The screening length controlled buckling of the ribbon suggests a convenient way in experiment and applications to manipulate ribbon morphologies by simply changing the salt concentration.
- We observed the dynamics of generic n-point vacancies in two-dimensional Lennard-Jones crystals in several thermodynamic states. We numerically observe significantly faster diffusion of the two-point vacancies that can be attributed to its rotational degree of freedom. The high mobility of the two-point vacancies opens the possibility of doping two-point vacancies into atomic materials to enhance atomic migration.
- We determine the rules for optimal self-replication of colloids. Using a quantitative analytic model, we correctly described the dependence of the dimer replication rate on the frequency of driving energy delivery, and the optimization of such frequency to minimize error during the replication process. We analyzed the self-replication of functionalized colloids. We also proposed a new paradigm based on magnetic colloids.

Summary:

Cells contain charged molecules, organized via reversible interactions into supramolecular structures and materials, which can perform mechanical and chemical functions with precision in fluctuating media. Our work aims to uncover the principles underlying the cellular programming responsible for compartmentalization, communication and replication as well as to subsequently employ these principles to generate assemblies that function as autonomous organisms. Living matter is highly inhomogeneous at the nano-scale both in composition and charge. A big challenge has been to determine how these inhomogeneities generate functions. In this context, the dominant role played by electrostatics during pattern formation is evident, given that nucleic acids and many proteins are charged or possess units with tunable degrees of charge. As a result, the association of biomolecules into functional units has demonstrated strong dependence on local ionic concentrations. Thus, using ionic gradients as one means to control the assembly and disassembly of biological entities is critically important to generate biomimetic functions. We have developed models, based on information collected *in vivo* and *in situ*, to better understand the physical properties and functions of biomolecular assemblies in different ionic conditions for designing biomimetic materials with new capabilities.

Build-up of our TARDIS cluster and a series of works on electrostatically driven structures and functions in multiple length scales carried out in the cluster

In Year 1 of the grant considerable effort was spent on acquiring and building the TARDIS cluster that was used for numerical simulations of the soft-matter systems proposed for study in the current fellowship. In technical details, the TARIS cluster has a total of **576 CPU cores** spread over **36 computational nodes** with **36 Fermi graphics cards**, one in each node. Nodes were connected through the **low-latency Infiniband** network, making it suitable for running large-scale, communication-intensive applications like molecular dynamics simulations or linear algebra calculations (e.g., in differential equation solvers).

To illustrate the computational power and efficiency of the TARDIS cluster, we have reported a GPU implementation in HOOMD Blue of long-range electrostatic interactions based on the orientation-averaged Ewald sum scheme, introduced by Yakub and Ronchi (*J. Chem. Phys.* **2003**, *119*, 11556). The performance of the method is compared to an optimized CPU version of the traditional Ewald sum available in LAMMPS, in the molecular dynamics of electrolytes. Our GPU implementation was found to be significantly faster than the CPU implementation of the Ewald method for small to a sizable number of particles ($\sim 10^5$). Thermodynamic and structural properties of monovalent and divalent hydrated salts in the bulk are calculated for a wide range of ionic concentrations. An excellent agreement between the two methods was found at the level of electrostatic energy, heat capacity, radial distribution functions, and integrated charge of the electrolytes.

As detailed in our publication in *Proceedings of the National Academy of Sciences USA*, we proposed a theoretical model that would provide a simple mechanism for the formation of general systems exhibiting the Platonic and Archimedean geometries found in many multicomponent elastic membranes. Large crystalline molecular shells, such as some viruses and fullerenes, buckle spontaneously into icosahedra. Meanwhile multicomponent microscopic shells buckle into various polyhedra, as observed in many organelles. Although elastic theory explains one-component icosahedral faceting, the possibility of buckling into other polyhedra has not

been explored. We showed that irregular and regular polyhedra, including some Archimedean and Platonic polyhedra, arise spontaneously in elastic shells formed by more than one component. By formulating a generalized elastic model for inhomogeneous shells, we demonstrated that co-assembled shells with two elastic components buckle into polyhedra such as dodecahedra, octahedra, tetrahedra, and hosohedra shells via a mechanism that explains many observations, predicts a new family of polyhedral shells, and provides the principles for designing micro-containers with specific shapes and symmetries for numerous applications in materials and life sciences.

In another study, we proposed a continuum theory to describe the influence of salt on the phase segregation behavior of polyelectrolyte gels. As polyelectrolyte (PE) gels are becoming more and more attractive materials for vast applications, it is of interest to develop a continuum theory to describe the influence of environmental stimuli on the phase behavior of PE gels. To quantitatively investigate the complex phase behavior, we employed generalized time-dependent Ginzburg-Landau evolution equations; in particular, the charge interactions were solved using a spectral technique that dramatically improves the computational efficiency. We identified the existence of nano-phases in between homogeneous swollen gels and collapsed gels. The characteristic length scale of PE gels in a bad solvent was investigated by taking the phase to be the stripe phase. Besides the numerical studies, we provide a weakly nonlinear analysis to shed a light on underlying physics that influences the characteristic length scale; in particular, we illustrated the salt concentration can be used to tune the length scale of complex patterns. The length scale of nano-phases can be adjusted by other system parameters, such as Bjerrum length, monomer charge fraction, etc (Wu *et al*., *Macromolecules* **43,** 9160 (2010).

In addition to the prediction of the length scale of the nano-phase, we investigated complex phase behavior in higher dimensions. PE gels are shown to exhibit different phases such as rolls, squares, hexagons, etc. The nano-patterns are sensitive to environmental parameters, and the phase transition can be achieved by continuously varying the environmental parameters. One example is the transition between hexagonal phase and reverse hexagonal phase accompanied by varying Bjerrum length or solvent quality. The asymmetry between these two hexagonal phases gives rise to asymmetry of solvent permeability, which can be used as solvent channel switch. Over the past decade it has become more and more obvious that the nuclear lamins are essential elements in determining nuclear structure and function. The lamins are responsible for determining nuclear shape and size, and they provide mechanical stability to the nucleus. Details of the molecular structure of lamins in the nucleus are not well understood. To investigate the morphology of bleb formation in cancer cell, we adapted a continuum approach, a phase field crystal model, which is capable of describing surface segregation, elastic energy and bending energy. A closest point method is used to map the evolution of phase field crystal equation to an arbitrary curved surface.

Continuing with our focus on understanding structures and functions, we investigated the surface patterns of charged end-group ligands attached to faceted nanoparticles using coarse-grained molecular dynamics simulations. A competition between electrostatic repulsion and hydrophobic ligand-ligand attraction leads to the formation of a number of different conformations of the ligand coatings. The most prominent conformation in icosahedral nanoparticles is a ridge-like structure that makes their surfaces highly anisotropic. Meanwhile, bundles seem more prominent than ridges for tetrahedral, cubic, octahedral, and dodecahedral

nanoparticles of diameters comparable to the chain length. The applicability of the Debye-Huckel theory to describe the ridges is confirmed by comparing simulations with explicit ion simulations. Here, we argued that a tunable ligand-coating pattern can be used as a simple and robust tool for achieving direction-dependent interactions between nanoparticles and provide control of their assembly into composite materials with a desired symmetry.

Furthermore, we also presented a model for the conformation of closed, inextensible diblock fibers. Each component of the fiber maintains a different stiffness. The relative fraction of each component is controlled by adding a chemical-potential term to the polymer's energy functional. We developed an analytical formalism that exploits variational arguments to derive the shape of two-component polymers for arbitrary component stiffness and length. These results yielded an analytical solution for the shape of diblock fibers in which each component has a different stiffness, a complete description of all possible polymer conformations and a phase portrait detailing the parameter spaces in which these shapes occur. Our work has developed a catalog of fiber conformations as a function of a variety of mechanical input conditions.

Electrostatic interactions are essential to designing self-assembled structures with unique symmetries and mechanical properties. These interactions can be modified by varying the effective charge of the molecules via the alteration of medium conditions including pH, salt concentration, or dielectric constant variations. In many situations, correlated ionic crystals form at liquid-liquid interfaces, on membrane surfaces, or at solid interfaces that adsorb charged molecules when such medium conditions are modified. We have determined the structure and mechanical properties, such as the Young's Modulus and Poisson's ratio, of stoichiometric 1:1-ionic crystals on surfaces as a function of the medium dielectric constant, which is modified by changing the solvent conditions in both theoretical calculations and simulations. We found that, at large values of the dielectric constant, the fully-packed hexagonal lattice has the lowest energy, while at small values of the dielectric constant the square lattice is the most favorable energetically. For square lattices, the Young's modulus along the direction of like-like charge particles is smaller than that along the direction of like-unlike charges, while its value remains isotropic in hexagonal crystals.

In general, polyvalent oligonucleotide-functionalized gold nanoparticles are remarkably stable in a cellular environment against degradation by nucleases, which was a property recently attributed to the local high concentration of mono- and divalent ions. In order to evaluate this hypothesis in more details, we investigated the composition of the ion cloud around spherical nanoparticles that are functionalized by stiff, highly charged polyelectrolyte chains by means of classical density functional theory and molecular dynamics simulations. As such, we presented a cell model that includes ligands explicitly and is both applicable over the entire relevant parameter space and in excellent quantitative agreement with simulations. We studied the ion cloud for varying oligonucleotides grafting densities, bulk ionic concentrations as well as the different sizes of nanoparticles and chains, and distinguish a parameter regime where many-body interactions between the ligands have a dominant effect on the local environment. For small particles with high oligonucleotide surface densities, we found strongly enhanced local salt concentrations, a large radial component of the electric field between the ligands, and a pronounced localization of divalent ions near the surface of the nanoparticle, thus providing multiple supporting arguments for the hypothesis.

In addition to statics, we also considered dynamics effects of non-equilibrium charge screening in mixtures of oppositely charged interacting molecules on surfaces in a closed system. The dynamics of charge screening and the strong deviation from the standard Debye-Huckel theory are demonstrated by computing radial distribution functions suited for analyzing both short-range and long-range spatial ordering effects with time. At long distances, the inhomogeneous molecular distribution is limited by diffusion, whereas at short distances by a balance of short-range (Lennard-Jones) and long-range (Coulomb) interactions. This generates different growth rates for the two characteristic length scales. The short length-scale saturates rapidly, while the long length-scale grows linearly with time.

Furthermore, we have developed a variational formulation of electrostatics that uses the ideal combination of the functional and the optimization scheme to allow for large-scale simulation of charges in the presence of varying dielectric response. At present, the most promising functionals and the accompanying numerical schemes employ an expensive choice of vector-function variables and/or make severe assumptions about the electrostatic system that they intend to simulate, such as the constraint on the charges to reside in only one dielectric media and restrictions on the geometry of the interface to simple shapes. We have formulated a variational principle for a general electrostatic problem with only one scalar basic variable function: the induced polarization charge density. The functional that we derive is a true energy functional, retaining its physical interpretation of energy, even outside the point of minimum, and yielding the correct thermodynamic force during the approach to the minimum, making the full exploitation of dynamical optimization procedures possible. Apart from the standard assumption of linear dielectric response, we make no assumptions about the electrostatic system in the derivation of the functional. In the specific problem of charges near one interface separating two dielectrics characterized by different dielectric constants, the induced charges reside only on the surface, thereby reducing the complexity of the problem to a two-dimensional one, and our computational scheme takes full advantage of this fact. The induced surface charge density can be expanded in terms of spherical harmonics for an interface of any general shape, the coefficients of expansion becoming the undetermined variational parameters that are solved for on the fly and in tandem with the production of charge configurations that become available to carry out statistical averages of quantities of interest. We have set up a computational scheme inspired by the dynamical optimization algorithm discovered by Car and Parrinello in the context of molecular dynamics simulation of ion-electron systems. In this sense, our method can be thought of as a unified approach to molecular dynamics and electrostatics, just as the Car-Parrinello method combines molecular dynamics with density functional theory. We have implemented our scheme to a model problem of a single charge near a dielectric sphere of a different dielectric constant when compared with that of the medium surrounding the charge. This model could be used to represent ions near a colloidal particle or, when the charges are inside, it could be used to study solvation effects on macromolecules. We have computed the polar profiles (which match exact results) of the induced surface charge density and the evolving position of the charge for both cases of the charge being outside and inside using both microcanonical and canonical simulations. Our next goal is to perform canonical simulations with many charges near interfaces of various shapes and extract physically important observables such as, for example, density profiles for the charges. This would require formulating the problem with the discrete analog of surface charge density and a convenient choice for the discretization of a general surface. We are very close to deriving and setting up such a formulation. A more ambitious long-term goal is to investigate the effects of evolving geometry,

which changes the boundary conditions that determine the electrostatic interaction between the ions, which in turn modifies the induced charge density on the interface and the very shape of the interface itself, thus inviting a deeper study of the interplay between geometry and electrostatic interactions.

As part of our ongoing works, we simulate a coarse-grained model of a system of proteins interacting with DNA via excluded volume interactions. In general, small proteins that bind to DNA play a significant role in controlling chromosome function and dynamics. The dynamics of DNA-binding proteins finding their targets has been a subject of intense study. However, release of proteins from DNA is also very important in controlling the dynamics of gene regulation. The stability of the protein-DNA structure depends to some degree on the fact that the proteins can unbind and rebind very quickly to the double helix. These events depend on the microscopic and macroscopic off-rates. We investigate how the DNA's conformation affects the protein rebinding dynamics. In the case of a stretched DNA, we find a logarithmic dependence. We also study the case where the DNA has a random self-avoiding walk shape: i) when it's free in space and ii) when it's confined in a sphere. In both these cases we find power-law dependence (the number of protein rebinding events vs. DNA length). Next we are going to include the appropriate charge in our system and study the same quantity. That is to see how the DNA conformation affects the binding-rebinding dynamics of the sequence-independent proteins binding to it.

Examining the shape space of multicomponent elastic membranes, electrically charged and DNA programmed nanoparticles, and fluid of charged hard spheres.

During the grant we effectively used and also expanded the TARDIS cluster, built with a unique design which allows for concurrent execution of conventional, highly-parallel CPU codes and novel GPU programs in a homogeneous environment and even within the same queuing system. The TARDIS cluster has continued to run at or near capacity this year, with all 576 CPU cores and 36 GPUs assembled in year 1 performing well. Users make use of a unified job-submission structure for serial, large-scale parallel, and GPU jobs. Upgrades to keep software and drivers current have been performed online with no resulting downtime. We have purchased and added a 12TB dedicated storage server using RAID-6 to accommodate large data sets and keep nightly backups; this server provides a secure place to store critical information and assists in the data-management side of achieving data provenance. Additionally, the success of running large-scale simulations on GPUs has motivated the recent purchase of 6 new computational nodes with Sandy Bridge Xeon processors that accommodate 5 GPUs each. This brought the group total to 66 GPUs.

In continuation of our earlier work describing a theoretical model that would provide a simple mechanism for the formation of general systems exhibiting the Platonic and Archimedean geometries found in many multicomponent elastic membranes, we have used simulated annealing Monte Carlo simulations to determine optimal shapes and domain distributions of a two-component elastic shell over a wide range of inter-component line tensions, relative concentrations of the components, as well as relative bending rigidities. We found a rich gallery of shapes, including a number of polyhedral structures at moderate values of the line tension and the bending rigidity ratio. For large values of the line tension, we observed a complete segregation of the components into shells with two distinct sides, one predominantly spherical

and the other buckled, see Figure 2.

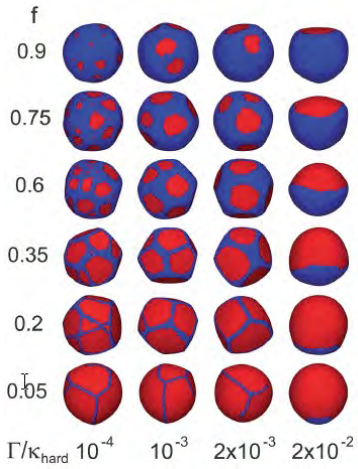


Figure 2: Gallery of representative shapes for a range of relative fractions of the soft component f vs. the dimensionless line tension Γ/κ_{hard} for $\kappa_{soft}/\kappa_{hard}$= 0.03. In the top row, the system is almost completely soft, and the icosahedral buckling is clearly evident. As the fraction of the soft component is reduced the icosahedral symmetry is lost, the soft component forms ridges and the shell buckles into a number of irregular polyhedra. The soft component is shown in blue and the hard component in red.

We argue that the observed faceting is driven by a mechanism very different from the one responsible for the buckling of a sphere into an icosahedron at the 12 disclination defects. For moderate values of the line tension the minority soft component forms lines, which act as edges of a polyhedron. The hard component segregates into facets that are flattened and the majority of the curvature is concentrated at the soft edges. Interesting shapes are observed for very weak line tension, in particular for values significantly smaller than the line tension used in a recent study of the multicomponent liquid vesicles. We believe this is due to the fact that even without the line tension, two elastic components on a shell naturally try to segregate, as we have recently reported. Thus, even a very weak line tension is sufficient to drive the components to fully segregate. Our analysis shows that it is in principle possible to combine two materials with different elastic properties to construct a wide variety of shell shapes with potential applications in medical imaging, drug delivery, or composite material design. In addition, results of this study provide a plausible explanation for the shapes of cell organelles such as carboxysomes or shapes obtained via coassembly of cationic and anionic amphiphiles. This study complements and considerably extends our recent analysis of shapes of the two-component shells in the absence of line tension. Our model assumes the presence of the 12 disclination defects originating from the construction of a crystal lattice on a sphere. However, our results suggest that the faceting into polyhedral shapes is not directly related to the defects, even though their presence introduces strain to the shell. It would be interesting to investigate the extent to which the defect-induced strain influences the faceting. In addition, it would be useful to develop a simple analytical model that would provide some scaling arguments for the edge length and width as a function of the line tension and bending rigidity. This work was featured on the cover of Soft Matter [1].

By combining molecular dynamics simulations and analytical arguments we investigated the elastic properties of charged lipid bilayers. We show that charge correlations between the head groups can lead to solidification of a lipid bilayer that would be in a liquid state if the charges were absent. All elastic parameters of the bilayer such as the bending rigidity κ and the two-dimensional Young's modulus Y are found to depend on the relative charge stoichiometric ratio between the lipid head groups. To extract κ we fit the molecular dynamics data to a standard elastic model for lipid bilayers. Moreover, we obtain the dependence of the Young's modulus on different charge patterns analytically by using Ewald summation techniques [10]. Furthermore, we used an elastic model to explore faceting of solid-wall vesicles with elastic heterogeneities. We show that faceting occurs in regions where the vesicle wall is softer, such as areas of reduced wall thicknesses or concentrated in crystalline defects. The elastic heterogeneities are modeled as a second component with reduced elastic parameters. Using simulated annealing Monte Carlo simulations we obtain the vesicle shape by optimizing the distributions of facets and boundaries. Our model allows us to reduce the effects of the residual stress generated by crystalline defects, and reveals a robust faceting mechanism into polyhedra other than the icosahedron [11].

We also studied the mean force and the potential of mean force between two identical charged nanoparticles immersed in a size-asymmetric monovalent electrolyte, showing that these results go beyond the standard description provided by the well-known Derjaguin-Landau-Verwey-Overbeek theory. To include consistently the ion-size effects, molecular dynamics (MD) simulations and liquid theory calculations are performed at the McMillan-Mayer level of description in which the solvent is taken into account implicitly as a background continuum with the suitable dielectric constant, see Figure 3.

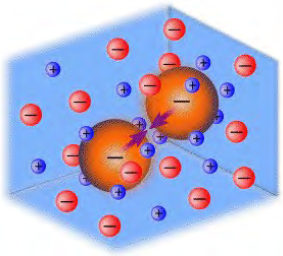


Figure 3: Attractive forces between two identical charged nanoparticles are observed in presence of a monovalent salt with different ionic size.

Long-range electrostatic interactions are handled properly in the simulations via the well-established Ewald sums method and the pre-averaged Ewald sums approach, originally proposed for homogeneous ionic fluids. An asymmetric behavior with respect to the colloidal charge polarity is found for the effective interactions between two identical nanoparticles. In particular, short-range attractions are observed between two equally charged nanoparticles, even though our model does not include specific interactions; these attractions are greatly enhanced for anionic nanoparticles immersed in standard electrolytes where cations are smaller than anions [2].

Solutions with highly charged solutes, thermal fluids of charged particles, and dense plasmas show an immensely rich phase behavior, all due to the dominant presence of the one fundamental interaction that sustains the entire complexity of life, the electromagnetic force. The structural and thermodynamic properties of such systems, and the mechanic response on external

stimuli, remain poorly understood, not merely because of the absence of one general predictive theoretical framework, but also due to the lack of clear interpretations of the results that follow from experimental observations and theoretical models. Current popular approaches are severely limited by the underlying assumptions that formally restrict the applicability to weakly coupled systems, i.e. where one has to preclude high charge concentrations, self-assembly, cluster formation, or any significant structure in the solvent at all. Correction methods for highly charged surfaces and interfaces focus on a thin layer near the boundary, or propose so-called loop expansions of virial expansions for the bulk behavior that, although aesthetic from a mathematical point of view, do not render these theories fit for general practical purposes, nor valid for the coupled systems that are relevant and ubiquitous in biology, soft matter, and applications. During the grant, a selection of popular methods were tested, and investigated for improvement, applied to study ions near flat dielectric boundaries between two immiscible electrolyte solutions, and to study effective interactions between charged, responsive surfaces. Within a density functional theory with a weighted density approximation (DFT/WDA), we tested several weight functions, and searched for consistent ways to derive them. Synchrotron X-ray reflectivity measurements are available for reference, thanks to a collaboration, and simulation methods are developed within the group to test the reliability of the new models down to a microscopic level. A choice was finally made for a less well-known approach that was initially proposed for the study of ions between two walls, including polarization and charge correlation effects. The approach relied on integral equation methods that are well-known and highly successful for homogeneous systems, but numerically intractable for general inhomogeneous systems. By exploiting symmetries, one can formally map the two-component inhomogeneous system to a polydisperse system of lower dimension that, although more complex in composition, is homogeneous, and hence can be tackled by these integral equation methods. We find a physical regime where these interactions easily overcome the thermal energy. The range is predicted to be non-monotonic with bulk concentration, and both the range and strength are found to depend on the confinement, i.e. dependent on the presence of other charged interfaces.

A central role in the determination of the structural and thermodynamic properties of a fluid is played by the pair correlation functions, and in the case of homogeneous isotropic fluids, the radial distribution function g(r). On the one hand, the radial distribution function is connected to the structure factor S(k), which is directly obtainable from scattering experiments, and on the other hand, to the thermodynamic properties, such as the internal energy, pressure, and compressibility. Thus, knowledge of the radial distribution function yields physical information both at the microscopic and macroscopic length scales. In principle, the radial distribution function can be obtained from the Ornstein-Zernike (OZ) equation, where it is related to the direct correlation function c(r), once an appropriate closure relation can be formulated. Despite the existence of an exact closure, the actual calculation requires approximative closures, whose accuracy depends on the character of the pair potentials between the particles of interest. We presented a method for the calculation of the radial distribution function g(r) in homogeneous fluids of charged, hard spheres,that aims to be accurate, consistent, practical, and applicable for all relevant temperatures and densities. The nonlinear Poisson-Boltzmann equation is obtained from an approximate closure relation of the Ornstein-Zernike equation, connecting nonlinear Debye-Hückel theory with integral equation methods. The nlDH theory appears quantitative at low densities and all densities at high temperatures if corrected for hard sphere correlations, by assuming pairwise additivity for the potentials of mean force. The resulting “mean field” g(r) is

used to estimate the two-point direct correlation function, by means of an approach that we named after the theories it derives from, the Debye-Hückel extended mean spherical approximation, or DHEMSA. The method provides almost identical structural and thermodynamic information as molecular dynamics simulations, and other reliable approximations such as the hyper-netted chain closure in a large part of parameter space. Moreover, it is numerically stable in the entire regime of interest, in contrast to the HNC, and numerically more efficient [3].

With respect to nanoparticles, we worked on analyzing bulk diffusion and transport through hydrophobic nanochannels of nanoparticles (NPs) with different hydrophobic-hydrophilic patterns achieved by coating a fraction of the NP sites with positive or negative charges via explicit solvent molecular dynamics simulations, see Figure 4.

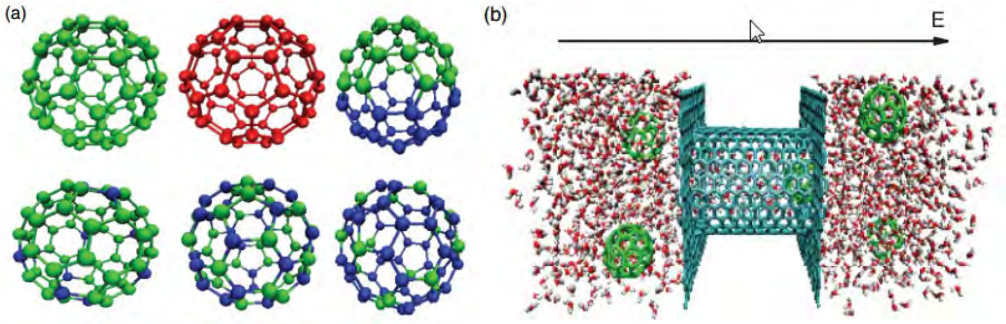


Figure 4: (a) Nanoparticle types: All have ±6e total charge. Green and red represent positive and negative charges, respectively, while blue denotes neutral, and (b) a snapshot of MD simulation system.

Ten different charge pattern types including Janus charged-hydrophobic NPs are studied. The cationic NPs are more affected by the patterns and have higher diffusion constants and fluxes than their anionic NPs counterparts. The NP-water interaction dependence on surface pattern and field strength explains these observations. The NP-water Coulomb interaction of anionic NPs in the bulk, which are much stronger than the hydrophobic NP-water interactions, are stronger for NPs with higher localized charge, and stronger than in the cationic NPs counterparts. The diffusion and transport of anionic NPs such as proteins and protein charge ladders with the same total charge but different surface charge patterns are slowest for the highest localized charge pattern, which also adsorb strongest onto surfaces. Our model demonstrates the separation (by reverse osmosis, capillary electrophoresis, or chromatography) of cationic NPs, including proteins with equal net charge but different surface charge distributions [4].

We studied how polyvalent oligonucleotide-functionalized gold nanoparticles are remarkably stable in a cellular environment against degradation by nucleases, a property that was recently attributed to the local high concentration of mono- and divalent ions. To evaluate this hypothesis in more detail, we investigated the composition of the ion cloud around spherical nanoparticles that are functionalized by stiff, highly charged polyelectrolyte chains by means of classical density functional theory and molecular dynamics simulations, as illustrated in Figure 5.

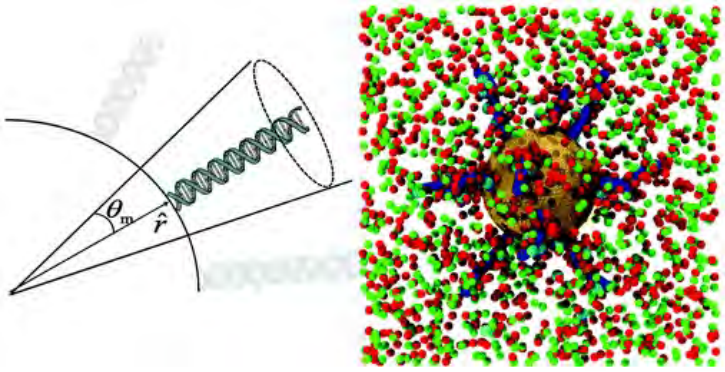


Figure 5: Cell model (left), and simulation snapshot (right) of the local ionic environment around a small polyvalent oligonucleotide-functionalized gold nanoparticle, with Na^+ (red), Cl^- (green), Ca^{2+} (cyan), and DNA (blue).

We presented a cell model that includes ligands explicitly and both applies over the entire relevant parameter space and is in excellent quantitative agreement with simulations. We studied the ion cloud for varying oligonucleotide grafting densities and bulk ionic concentrations, as well as different sizes of nanoparticles and chains, and distinguish a parameter regime where many-body interactions between the ligands have a dominant effect on the local environment. For small particles with high oligonucleotide surface densities, we find strongly enhanced local salt concentrations, a large radial component of the electric field between the ligands, and a pronounced localization of divalent ions near the surface of the nanoparticle, thus providing multiple supporting arguments for the hypothesis [5].

The strategy of using DNA to program the assembly of nanoparticles into macroscopic materials emerged in the mid-1990s. Over the past decade, substantial advances have been achieved in transforming this wet chemical technique from one that can be used to generate amorphous or pseudocrystalline architectures into one that yields highly crystalline materials with a high degree of predictability in terms of crystal type and lattice parameters. With this bottom-up approach, spherical nucleic acid gold nanoparticle conjugates (SNA-GNPs) can be used as artificial "atoms", where the oligonucleotides connecting the SNA-GNPs are “chemical bonds” that can be used to create novel one-, two-, and three-dimensional superlattices. In previous work, six rules for predicting superlattice structure have been developed. These rules take into account the nanoparticle size, DNA length, overall number of DNA connections, and sequence complementarity to predict and understand the periodic packing of nanoparticles in discrete lattice types with lattice constants in the range of tens to hundreds of nanometers. This degree of tailorability provides the opportunity to control the properties of crystalline nanoparticle architectures with potential applications in a variety of fields, including medical diagnostics, catalysis, energy conversion, and plasmonics. In collaboration with the group of Chad Mirkin, we used molecular dynamics simulations to study the crystallization of spherical nucleic-acid (SNA) gold nanoparticle conjugates, guided by sequence-specific DNA hybridization events. This is illustrated in Figure 6.

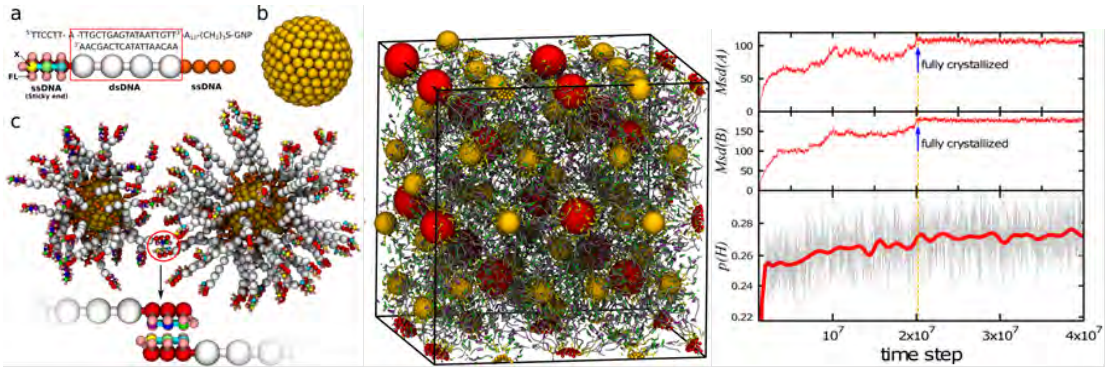


Figure 6: Left: Model DNA chain (a); model spherical GNP core (b); and SNA-GNP examples (c). Middle: Snapshot of crystal structure obtained by MD simulations. Right: Mean square displacement of larger (top) and smaller (middle) SNA-GNPs, and (bottom) percentage of hybridization versus simulation time step.

Binary mixtures of SNA gold nanoparticle conjugates (inorganic core diameter in the 8–15 nm range) are shown to assemble into BCC, CsCl, AlB_2, and Cr_3Si crystalline structures, depending upon particle stoichiometry, number of immobilized strands of DNA per particle, DNA sequence length, and hydrodynamic size ratio of the conjugates involved in crystallization. These data have been used to construct phase diagrams that are in excellent agreement with experimental data from wet-laboratory studies and the work was published in Nano Letters [9].

The stability of a DNA-protein structure depends, among other factors, on the fact that small proteins can unbind and rebind very quickly to the double helix. These events depend on the microscopic and macroscopic off-rates of the proteins. Generally a small protein will go through a number of unbinding-rebinding events before it diffuses off in its surrounding buffer. These events are directly associated with the affinity of the proteins to the DNA. We studied, in particular, how these events depend on the DNA's shape and conformation. These events play a significant role in controlling the dynamics of gene regulation. Our study was done by simulating a coarse-grained model of a system of sequence-independent proteins interacting with DNA. We used Monte-Carlo as our simulation method. In the first part of our study we considered only excluded volume interactions between the small proteins and the double helix. We investigated how the DNA's conformation affects the protein rebinding dynamics. We studied three general cases of i) a stretched DNA, ii) a random shaped DNA, and iii) a DNA with a globule structure confined in a sphere (where the spheres radius is less than the radius of gyration of the DNA polymer). To represent the microscopic and macroscopic off-rates, we consider two time scales associated with the protein dynamics. In the case of stretched DNA, we found a logarithmic dependence between the DNA's length and the rebinding events. In the case of a random shaped DNA, we found the two above quantities to be related in a power law form. We generated two different powers in the case of the unconfined and confined polymers. The power laws and the logarithmic form agree with our theoretical scaling arguments. In the second part of our study, we modified the system to include the charge of a specific sequence-independent protein (FIS) as well as the DNA charge. We included the Coulomb interaction to investigate how the DNA conformation affects the dynamics of the sequence-independent proteins binding to it, in the three cases mentioned above.

Combining theory and simulations to study charged fluids, lamin meshworks and structured membranes for elucidating structure and function relationship.

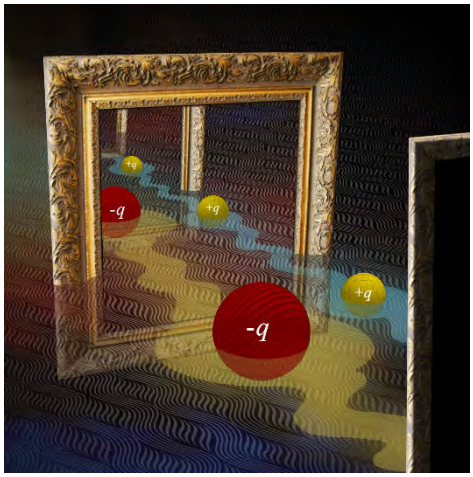


Figure 1: Charged particles in a liquid are enveloped by a cloud of opposite charge that deforms near boundaries, and creates a force away from the boundary. Electrostatically, boundaries act as mirrors, causing additional attractions or repulsions. Theory reveals how ions influence forces between boundaries in an intricate way, and vice versa, with important implications for biological systems (Artistic impression by D. Spjut).

Fluids of charged particles act as the supporting medium for chemical reactions and physical, dynamical, and biological processes. Micro- and nanoscopic polarizable objects deform the local structure in an electrolytic background. Vice versa, the forces between the objects are regulated by the cohesive properties of the background (Figure 1). We studied the range and strength of these forces and the microscopic origin from which they emerge [1], providing an intuitively appealing overview to aid and enhance predictability and control in experiments. We described several distinct ion-induced forces in aqueous solutions of monovalent ions at salt concentrations ranging from 0.01 to 1 M that are expected to compete for dominance with the well-known screened Coulomb forces. The concept of soft structure was introduced to connect the induced forces to the local microscopic structure in the ionic fluid and visualize the deformation of the local environment around the ions near several types of boundaries that are neutral, adsorbing, charged, polarizable, or a combination of these properties. Until recently, it was computationally costly to simulate the effects of charges on inhomogeneous dielectric media and vice versa, since it required solving the Poisson equation at each simulation step. In a related study, we developed a judiciously designed variational formulation, eliminating the need to solve the Poison equation explicitly.

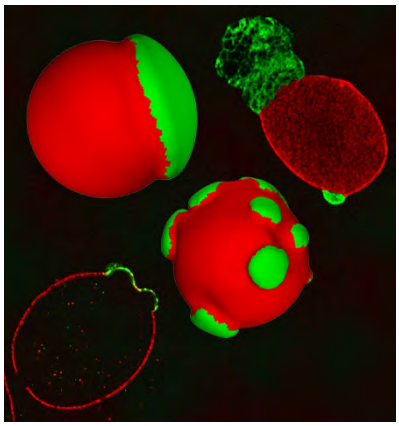


Figure 2: Experimental and computational images of nuclei showing the segregation of two types of lamin proteins in cell nuclei. A-type lamin is shown in green, while B-type is shown in red (Image modified from simulation snapshots and TEMs by M. Seniw).

Much of the structural stability of the cells's nucleus comes from meshworks of intermediate filament proteins known as lamins forming the inner layer of the nuclear envelope called the nuclear lamina. These lamin meshworks additionally play a role in gene expression. Abnormalities in nuclear shape are associated with a variety of pathologies, including some forms of cancer and Hutchinson–Gilford Progeria Syndrome, and often include protruding structures termed nuclear blebs. These nuclear blebs are related to pathological gene expression. However, little is known about how and why blebs form. We developed a minimal continuum elastic model of a lamin meshwork that we use to investigate which aspects of the meshwork could be responsible for bleb formation (Figure 2). The model produces structures with comparable morphologies and distributions of lamin types as in real pathological nuclei [2]. Thus, preventing this lamin segregation could be a route to prevent bleb formation, which could be used as a potential therapy for the pathologies associated with nuclear blebs.

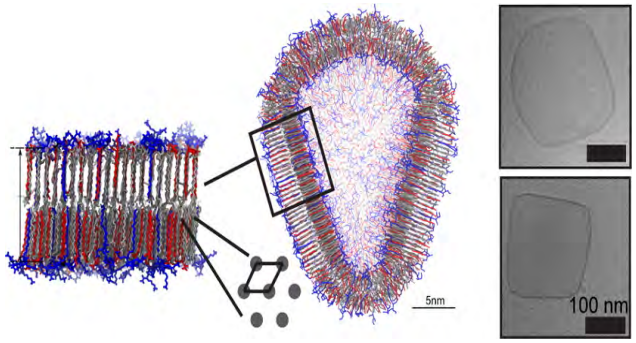


Figure 3: Atomistic simulations, and TEM images of faceted and irregular polyhedral with hexagonal crystalline symmetry, stable in closed shaped and high salt concentration.

A considerable part of our work has been aimed at elucidating structure and function relationship in membranes. A large variety of amphiphillic molecules with charged groups self-assemble into closed structures such as viral capsids, halophilic organisms and bacterial microcompartments. We formulated a generalized elastic model for inhomogeneous shells, and demonstrated that co-assembled shells with two elastic components buckle into irregular and regular polyhedra besides icosahedra, such as dodecahedra, octahedra, tetrahedra, and hosohedra shells via a mechanism

that explains many observations, predicts a new family of polyhedral shells, and provides the principles for designing micro-containers with specific shapes and symmetries for numerous applications in materials and life sciences [3]. For instance, designing polyhedral shells with specific structures can lead to efficient nano-reactors that will perform specific catalytic functions. In a collaborative effort with experimental collaborators, we modeled and assembled shells composed of oppositely charged lipids, and were able to regulate the shape of the resulting nanocontainers via pH changes, as evidenced by Transmission Electron Microscopy (TEM, Figure 3) and x-ray scattering [4]. Vesicles or nano-containers with crystalline properties show great promise in nanomaterial fabrication due to their excellent mechanical stability at high salt concentration environments. We also studied by atomistic simulations, the effects of counterion valency on anionic membranes. Our work showed that the inter-tail van der Waals interaction has a dominant role in the electrostatic driven transition from disordered liquid-crystalline phase to the ordered gel phase. Using monovalent counterions, such phase transitions occur at lower temperatures than in scenarios where multivalent counterions are used. Moreover, within the model of the Helfrich elasticity theory, we studied shapes of pored membranes, and show that the mean curvature leads to budding-like behavior, while the Gaussian curvature flattens membranes near the pore area. These findings were well confirmed in simulations. And our work also explains shapes of pored membranes that are observed in experiments.

Motivated by the remarkable stability against degradation by nucleases of functionalized spherical nuclei acids-gold (SNA-Au) nanoparticles in blood, we modeled the ion cloud around them by means of classical density functional theory and by computer simulations. For small particles with high density of oligonucleotides, we find that the local salt concentration is enhanced with a pronounced localization of divalent ions near the particle surface. This strong cloud of sodium and calcium ions stabilizes the particle complex in blood, which may prevent enzymes from accessing the DNA and digesting it. We also developed coarse-grained schemes to understand and optimize programmable assembly [5] of grafted nuclei acid-gold nanoparticles. In particular, our MD simulations have identified key elements, such as optimal strength of DNA linkers and percentage of DNA hybridizations, to successfully assemble nanoparticles into a large variety of superlattices. The MD simulations have also elucidated the role that nanostructure coordination plays in the ordering of superlattice assemblies. We constructed detailed phase diagrams that closely match the experimental observations, and showed that highly dynamic hybridization processes between the semi-flexible grafted chains enable crystallization of the functionalized nanoparticles. A model that efficiently captures the competing electrostatic and DNA hybridization interactions in the assembly process was developed to determine the dynamics of the coarsening process that leads to various faceted crystals of functionalized nanoparticles.

Monte Carlo Simulations for the Optimization of Self-replication

Understanding the key ingredients of self-replication is critical for developing self-sustaining systems in the laboratory. We proposed a scheme for self-replication where asymmetric interactions in colloids are used to find optimal self-replication conditions by controlling the input of energy. We generalized a recently developed kinetic Monte Carlo algorithm to treat both translational and rotational motions of Brownian anisotropic colloids. We found two main findings from our simulations: First, by fine-tuning the particle interactions, highly accurate self-replication is achievable with a moderate sacrifice of reaction speed. Second, with the

introduction of energy cycling to enable periodic assembly/disassembly of the system's components the replicator population grows exponentially. The exponential growth constant is a non-monotonic function of the period of the pulsed energy delivery.

Self-replication also permits organisms to reproduce and sustain complex cellular functions. These desirable properties have motivated extensive interdisciplinary efforts to develop artificial self-replication. While molecular self-replication has been extensively explored, colloidal self-replication is less developed despite its promise for unique functionality. One major obstacle in colloidal self-replication is that cyclic energy is required for efficient self-replication. We showed that magnetic colloids driven by magnetic fields can create an efficient self-replication system. Single binding sites allow the formation of colloid dimers, which serve as replication templates. Magnetic fields are used to drive complementary elements from the solute to bind to templates and to separate templates from each other. We show via theory and simulations that this system exhibits exponential growth while making minimal demands with regards to synthesis. Using a magnetic basis allows this system to operate in complex environments such as living tissue.

Understanding dielectric heterogeneities and ion correlations in the organization principle of ion distribution, polymer electrolytes, and the coupling of electrostatics and geometry in charged membranes and ribbons.

Small nanoparticles, globular proteins, viral capsids, and other nanoscopic biomolecules usually display dielectric properties that are different from those of the medium in which they are dispersed. These dielectric heterogeneities can significantly influence the surrounding ion distribution, which determines the self-assembly and colloidal stability of these nanoparticles in solution. We studied the impact of a dielectric discontinuity in the structural and thermodynamic properties of a spherical nanoparticle made of different dielectric materials when it is immersed in a charge-asymmetric 1:z supporting electrolyte. The mean electrostatic potential, integrated charge, and ionic profiles were analyzed as a function of both the salt concentration and the nanoparticle's valence via Monte Carlo simulations and the nonlinear Poisson-Boltzmann theory. We observed that the electrostatic screening and charge neutralization near the surface of a nanoparticle increase when the nanoparticle's dielectric permittivity increases in all instances. For 1:1 salts, this effect is small and the nonlinear Poisson-Boltzmann theory displays a good agreement with simulation results. Nevertheless, significant deviations are displayed by the mean field scheme regarding simulation results in the presence of multivalent ions. In particular, for trivalent counterions we observed that increasing the dielectric permittivity or the valence of the nanoparticle decreases the critical salt concentration at which a sign inversion of the mean electrostatic potential at the Helmholtz plane occurs. This is closely related to the behavior of the zeta potential and the electrophoretic mobility. Moreover, we observed that the phenomenon of surface charge amplification, or the augmenting of the net charge of a nanoparticle by the adsorption of like-charged ions on its surface, can be promoted by polarization effects in weakly charged spherical nanoparticles with low dielectric permittivity.

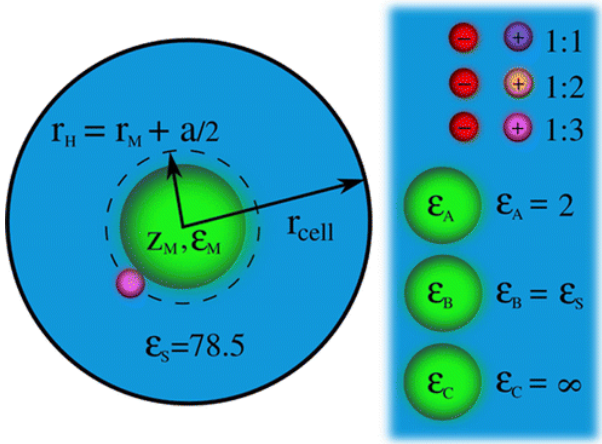


Figure1: Schematic representation of the model system. The dielectric constant at the interior of the nanoparticle is εM. The dielectric constant outside the nanoparticle, εS, is the same for the ions and the solvent.

We also analyzed polymer electrolytes. Polymer electrolytes, or polyelectrolytes, combine mechanical flexibility and highly tunable electrochemical properties, and therefore are promising candidate materials for energy storage devices, such as polymer electrolyte membrane fuel cells (PEMFC) and rechargeable Lithium-ion (Li-ion) batteries. The phase behavior of polyelectrolytes is especially important, as this dictates the formation of nanostructures within the material that guides ion transport. Thus, in order to optimize transport of ions through the material and increase efficiency of energy storage devices, it is crucial to develop predictive tools that can describe and explain the complex behavior of polyelectrolytes. We have developed a hybrid self-consistent field and liquid state theories (SCFT-LS) to incorporate ionic correlations at multiple length scales, leading to an accurate description of phases formed by polyelectrolyte blends and copolymers. To date, we have shown that the addition of ionic correlations can lead to enhanced phase separation and the formation of ion clusters. Furthermore, we have identified a triple point in an effectively two-component system and a eutectic-like transition into a miscible region. We believe that our findings can further inform synthesis and design of miscible polyelectrolyte blends toward energy storage applications.

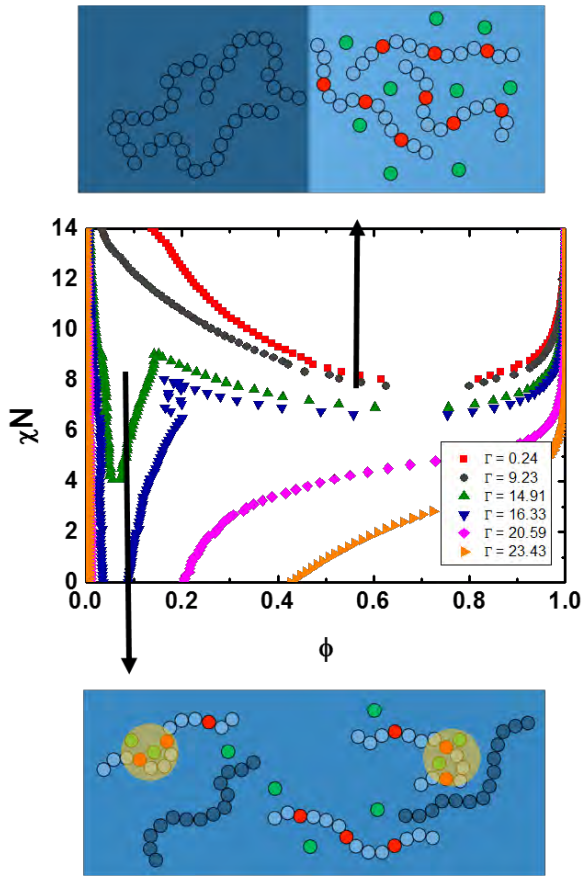


Figure 2. Phase diagram derived from free energy analysis for a symmetric polymer with number of monomers N = 40 and farction of charge f_q = 0.2, at varying values of the ionic coupling parameter in the medium Γ. Enhanced phase separation is observed at Γ > 14.91. The coexistence region splits into two distinct regions at Γ = 14.91 and Γ = 16.33, where the splitting point corresponds to the triple point, where the system undergoes a eutectic-like transition from a mixed state to two distinctive phases.

Manipulating the shape of nanoscale objects in a controllable fashion is at the heart of designing materials that act as building blocks for self-assembly or serve as targeted drug delivery carriers. Inducing shape deformations by controlling external parameters is also an important way of designing biomimetic membranes. We demonstrated that electrostatics can be used as a tool to manipulate the shape of soft, closed membranes by tuning environmental conditions such as the electrolyte concentration in the medium. Using a molecular dynamics-based simulated annealing procedure, we investigated charged elastic shells that do not exchange material with their environment, such as elastic membranes formed in emulsions or synthetic nanocontainers. We found that by decreasing the salt concentration or increasing the total charge on the shell's surface, the spherical symmetry is broken, leading to the formation of ellipsoids, discs, and bowls. Shape changes are accompanied by a significant lowering of the electrostatic energy and a rise in the surface area of the shell. To substantiate our simulation findings, we showed analytically that a uniformly charged disc has a lower Coulomb energy than a sphere of the same volume. Further, we tested the robustness of our results by including the effects of charge

renormalization in the analysis of the shape transitions and find the latter to be feasible for a wide range of shell volume fractions.

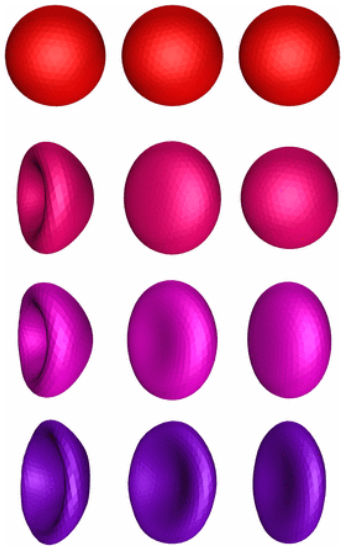


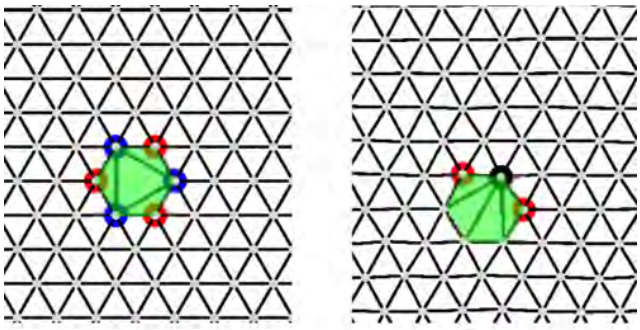
Figure 3. Snapshots of minimum-energy conformations of charged elastic nanoshells for three different bending rigidities κ = 1, 5, and 10 (columns from left to right). In each column the electrolyte concentration c(M) decreases (rows from top to bottom) as c = 1, 0.1, 0.05, and 0.005. Different colors suggest different concentration values, with red being the highest c under study and purple corresponding to the lowest c. As the concentration is lowered, the range of electrostatic interactions is increased, leading to the variation in the shape of the nanoshell. For the concentration range investigated, softer shells form bowl-like structures, wheras more rigid vesicles form ellipsoidal and disc-like shapes. All of the above nanostructures have the same total surface charge and volume, fixed to values associated with the spherical conformation.

We provided analytical expressions for the electrostatic energy of uniformly charged prolate and oblate spheroidal shells. We found that uniformly charged prolate spheroids of eccentricity greater than 0.9 have lower Coulomb energy than a sphere of the same area. For the volume-constrained case, we found that a sphere has the highest Coulomb energy among all spheroidal shells. Further, we derived the change in the Coulomb energy of a uniformly charged shell due to small, area-conserving perturbations on the spherical shape. Our perturbation calculations showed that buckling-type deformations on a sphere can lower the Coulomb energy. Finally, we considered the possibility of counterion condensation on the spheroidal shell surface. We employed the Manning-Oosawa two-state model approximation to evaluate the renormalized surface charge and analyze the behavior of the equilibrium free energy as a function of the shell's aspect ratio for both area-constrained and volume-constrained cases. Counterion condensation favors the formation of spheroidal structures over a sphere of equal area for high values of shell volume fractions.

With the combination of numerical simulations and mathematical analysis, we extended the electrostatic analysis from charged vesicles to charged ribbons which were recently synthesized in experiments with promising applications in realizing several biological functions and designing new structures in nanotechnologies. We compiled codes to perform numerical

experiments and investigate the rich morphologies of electrically charged flexible ribbon immersed in electrolyte solutions. Simulations show the hierarchical buckling of the ribbon from its initially flat shape to the undulated and then to the out-of-plane twisted conformations with the variation of salt concentration. We used the tool of classical differential geometry to analyze this conformal transformation and revealed that the deformation of the ribbon originates from the electrostatically driven stretching that fundamentally changes the metric distribution over the ribbon surface. In this study, we also identified the basic modes adopted by the ribbon to reshape itself for lowering the energy. The screening length controlled buckling of the ribbon suggests a convenient way in experiment and applications to manipulate ribbon morphologies by simply changing the salt concentration.

In soft crystals, behaviors of topological defects are strongly analogous to electric charges. Vacancies represent an important class of defects, and their behaviors can be strongly coupled with relevant material properties. A vacancy can be regarded as a compound topological defect that is composed of several topologically charged disclinations. In this work, we used LAMMPS to perform MD simulations to systematically study the dynamics of generic n-point vacancies in two-dimensional Lennard-Jones crystals in several thermodynamic states. We focused on the NPT emsemble with zero external pressure, while the NAT (fixed number of particles, system area and temperature) ensemble was employed to study the dependence of the vacancy mobility on the particle density. Simulations revealed the spectrum of distinct, size-dependent vacancy dynamics, including the nonmonotonously varying diffusive mobilities of one-, two- and three-point vacancies, and several healing routines of linear vacancies. Specifically, we numerically observed significantly faster diffusion of the two-point vacancy that can be attributed to its rotational degree of freedom. The high mobility of the two-point vacancies opens the possibility of doping two-point vacancies into atomic materials to enhance atomic migration. The rich physics of vacancies revealed in this study have implications in the engineering of defects in extensive crystalline materials for desired properties.



The topological defect structure associated with the one-point vacancy in a perfect hexagonal lattice (a) and at finite temperature (b). The red, blue, and black dots represent five-, seven-, and eight-fold disclinations, respectively. A slight fluctuation of the particles gives rise to distinct defect motifs. The region of the vacancy is shadowed in green.

Publications (partially or fully funded by this project, 2010-2015):

1. Guerrero-Garcia GI, Gonzalez-Mozuelos P, de la Cruz MO. Potential of mean force between identical charged nanoparticles immersed in a size-asymmetric monovalent electrolyte. Journal of Chemical Physics. 2011;135(16).
2. Guo PJ, Sknepnek R, de la Cruz MO. Electrostatic-Driven Ridge Formation on Nanoparticles Coated with Charged End-Group Ligands. Journal of Physical Chemistry C. 2011;115(14):6484-90.
3. Jha PK, Zwanikken JW, Detcheverry FA, de Pablo JJ, de la Cruz MO. Study of volume phase transitions in polymeric nanogels by theoretically informed coarse-grained simulations. Soft Matter. 2011;7(13):5965-75.
4. Kuzovkov VN, Kotomin EA, de la Cruz MO. The non-equilibrium charge screening effects in diffusion-driven systems with pattern formation. Journal of Chemical Physics. 2011;135(3).
5. Vernizzi G, Sknepnek R, de la Cruz MO. Platonic and Archimedean geometries in multicomponent elastic membranes. Proceedings of the National Academy of Sciences of the United States of America. 2011;108(11):4292-6.
6. Vernizzi G, Zhang DS, de la Cruz MO. Structural phase transitions and mechanical properties of binary ionic colloidal crystals at interfaces. Soft Matter. 2011;7(13):6285-93.
7. Zwanikken JW, Guo PJ, Mirkin CA, de la Cruz MO. Local Ionic Environment around Polyvalent Nucleic Acid-Functionalized Nanoparticles. Journal of Physical Chemistry C. 2011;115(33):16368-73.
8. Zwanikken JW, Jha PK, de la Cruz MO. A practical integral equation for the structure and thermodynamics of hard sphere Coulomb fluids. Journal of Chemical Physics. 2011;135(6).
9. Demers MF, Sknepnek R, de la Cruz MO. Curvature-driven effective attraction in multicomponent membranes. Physical Review E. 2012;86(2).
10. Jadhao V, Solis FJ, de la Cruz MO. Simulation of Charged Systems in Heterogeneous Dielectric Media via a True Energy Functional. Physical Review Letters. 2012;109(22).
11. Leung CY, Palmer LC, Qiao BF, Kewalramani S, Sknepnek R, Newcomb CJ, et al. Molecular Crystallization Controlled by pH Regulates Mesoscopic Membrane Morphology. Acs Nano. 2012;6(12):10901-9.
12. Li T, Sknepnek R, Macfarlane RJ, Mirkin CA, de la Cruz MO. Modeling the Crystallization of Spherical Nucleic Acid Nanoparticle Conjugates with Molecular Dynamics Simulations. Nano Letters. 2012;12(5):2509-14.
13. Sknepnek R, Vernizzi G, de la Cruz MO. Charge renormalization of bilayer elastic properties. Journal of Chemical Physics. 2012;137(10).
14. Sknepnek R, Vernizzi G, de la Cruz MO. Buckling of multicomponent elastic shells with line tension. Soft Matter. 2012;8(3):636-44.
15. Su JY, de la Cruz MO, Guo HX. Solubility and transport of cationic and anionic patterned nanoparticles. Physical Review E. 2012;85(1).
16. Yao ZW, Sknepnek R, Thomas CK, de la Cruz MO. Shapes of pored membranes. Soft Matter. 2012;8(46):11613-9.
17. Ziebert F, Swaminathan S, Aranson IS. Model for self-polarization and motility of keratocyte fragments. Journal of the Royal Society Interface. 2012;9(70):1084-92.
18. Funkhouser CM, Sknepnek R, de la Cruz MO. Topological defects in the buckling of elastic membranes. Soft Matter. 2013;9(1):60-8.

19. Funkhouser CM, Sknepnek R, Shimi T, Goldman AE, Goldman RD, de la Cruz MO. Mechanical model of blebbing in nuclear lamin meshworks. Proceedings of the National Academy of Sciences of the United States of America. 2013;110(9):3248-53.
20. Gonzalez-Mozuelos P, Guerrero-Garcia GI, de la Cruz MO. An exact method to obtain effective electrostatic interactions from computer simulations: The case of effective charge amplification. Journal of Chemical Physics. 2013;139(6).
21. Guerrero-Garcia GI, de la Cruz MO. Inversion of the Electric Field at the Electrified Liquid-Liquid Interface. Journal of Chemical Theory and Computation. 2013;9(1):1-7.
22. Guerrero-Garcia GI, Gonzalez-Mozuelos P, de la Cruz MO. Large Counterions Boost the Solubility and Renormalized Charge of Suspended Nanoparticles. Acs Nano. 2013;7(11):9714-23.
23. Guerrero-Garcia GI, Jing YF, de la Cruz MO. Enhancing and reversing the electric field at the oil-water interface with size-asymmetric monovalent ions. Soft Matter. 2013;9(26):6046-52.
24. Jadhao V, Solis FJ, de la Cruz MO. Free-energy functionals of the electrostatic potential for Poisson-Boltzmann theory. Physical Review E. 2013;88(2).
25. Jadhao V, Solis FJ, de la Cruz MO. A variational formulation of electrostatics in a medium with spatially varying dielectric permittivity. Journal of Chemical Physics. 2013;138(5).
26. Kewalramani S, Zwanikken JW, Macfarlane RJ, Leung CY, de la Cruz MO, Mirkin CA, et al. Counterion Distribution Surrounding Spherical Nucleic Acid-Au Nanoparticle Conjugates Probed by Small-Angle X-ray Scattering. Acs Nano. 2013;7(12):11301-9.
27. Leung CY, Palmer LC, Kewalramani S, Qiao BF, Stupp SI, de la Cruz MO, et al. Crystalline polymorphism induced by charge regulation in ionic membranes. Proceedings of the National Academy of Sciences of the United States of America. 2013;110(41):16309-14.
28. Li T, Sknepnek R, de la Cruz MO. Thermally Active Hybridization Drives the Crystallization of DNA-Functionalized Nanoparticles. Journal of the American Chemical Society. 2013;135(23):8535-41.
29. Parsaeian A, de la Cruz MO, Marko JF. Binding-rebinding dynamics of proteins interacting nonspecifically with a long DNA molecule. Physical Review E. 2013;88(4).
30. Qiao BF, de la Cruz MO. Driving Force for Water Permeation Across Lipid Membranes. Journal of Physical Chemistry Letters. 2013;4(19):3233-7.
31. Qiao BF, de la Cruz MO. Driving Force for Crystallization of Anionic Lipid Membranes Revealed by Atomistic Simulations. Journal of Physical Chemistry B. 2013;117(17):5073-80.
32. Sing CE, Zwanikken JW, de la Cruz MO. Interfacial Behavior in Polyelectrolyte Blends: Hybrid Liquid-State Integral Equation and Self-Consistent Field Theory Study. Physical Review Letters. 2013;111(16).
33. Sing CE, Zwanikken JW, de la Cruz MO. Effect of Ion-Ion Correlations on Polyelectrolyte Gel Collapse and Reentrant Swelling. Macromolecules. 2013;46(12):5053-65.
34. Sing CE, Zwanilden JW, de la Cruz MO. Ion Correlation-Induced Phase Separation in Polyelectrolyte Blends. Acs Macro Letters. 2013;2(11):1042-6.
35. Solis FJ, Jadhao V, de la Cruz MO. Generating true minima in constrained variational formulations via modified Lagrange multipliers. Physical Review E. 2013;88(5).

36. Thomas CK, de la Cruz MO. Theory and simulations of crystalline control via salinity and pH in ionizable membranes. Soft Matter. 2013;9(2):429-34.
37. Yao ZW, de la Cruz MO. Topological Defects in Flat Geometry: The Role of Density Inhomogeneity. Physical Review Letters. 2013;111(11).
38. Yao ZW, de la Cruz MO. Electrostatic repulsion-driven crystallization model arising from filament networks. Physical Review E. 2013;87(4).
39. Yao ZW, de la Cruz MO. Packing of charged chains on toroidal geometries. Physical Review E. 2013;87(1).
40. Yao ZW, Qiao BF, de la Cruz MO. Potassium ions in the cavity of a KcsA channel model. Physical Review E. 2013;88(6).
41. Zhang R, Jha PK, de la Cruz MO. Non-equilibrium ionic assemblies of oppositely charged nanoparticles. Soft Matter. 2013;9(20):5042-51.
42. Zwanikken JW, de la Cruz MO. Tunable soft structure in charged fluids confined by dielectric interfaces. Proceedings of the National Academy of Sciences of the United States of America. 2013;110(14):5301-8.
43. Auyeung E, Li T, Senesi AJ, Schmucker AL, Pals BC, de la Cruz MO, et al. DNA-mediated nanoparticle crystallization into Wulff polyhedra. Nature. 2014;505(7481):73-7.
44. Garcia GIG, de la Cruz MO. Polarization Effects of Dielectric Nanoparticles in Aqueous Charge-Asymmetric Electrolytes. Journal of Physical Chemistry B. 2014;118(29):8854-62.
45. Jadhao V, Thomas CK, de la Cruz MO. Electrostatics-driven shape transitions in soft shells. Proceedings of the National Academy of Sciences of the United States of America. 2014;111(35):12673-8.
46. Ovanesyan Z, Medasani B, Fenley MO, Guerrero-Garcia GI, de la Cruz MO, Marucho M. Excluded volume and ion-ion correlation effects on the ionic atmosphere around B-DNA: Theory, simulations, and experiments. Journal of Chemical Physics. 2014;141(22).
47. Qiao B, Demars T, de la Cruz MO, Ellis RJ. How Hydrogen Bonds Affect the Growth of Reverse Micelles around Coordinating Metal Ions. Journal of Physical Chemistry Letters. 2014;5(8):1440-4.
48. Sing CE, de la Cruz MO. Polyelectrolyte Blends and Nontrivial Behavior in Effective Flory-Huggins Parameters. Acs Macro Letters. 2014;3(8):698-702.
49. Sing CE, de la Cruz MO, Marko JF. Multiple-binding-site mechanism explains concentration-dependent unbinding rates of DNA-binding proteins. Nucleic Acids Research. 2014;42(6):3783-91.
50. Yao ZW, de la Cruz MO. Dynamics of vacancies in two-dimensional Lennard-Jones crystals. Physical Review E. 2014;90(6).
51. Yao ZW, de La Cruz MO. Polydispersity-driven topological defects as order-restoring excitations. Proceedings of the National Academy of Sciences of the United States of America. 2014;111(14):5094-9.
52. Zhang R, Dempster JM, de la Cruz MO. Self-replication in colloids with asymmetric interactions. Soft Matter. 2014;10(9):1315-9.
53. Boon N, Guerrero-Garcia GI, van Roij R, de la Cruz MO. Effective charges and virial pressure of concentrated macroion solutions. Proceedings of the National Academy of Sciences of the United States of America. 2015;112(30):9242-6.
54. Dempster JM, Zhang R, de la Cruz MO. Self-replication with magnetic dipolar colloids. Physical Review E. 2015;92(4).

55. Jadhao V, Yao ZW, Thomas CK, de la Cruz MO. Coulomb energy of uniformly charged spheroidal shell systems. Physical Review E. 2015;91(3).
56. Kwon HK, Zwanikken JW, Shull KR, de la Cruz MO. Theoretical Analysis of Multiple Phase Coexistence in Polyelectrolyte Blends. Macromolecules. 2015;48(16):6008-15.

Personnel

Faculty:	Prof. Monica Olvera de la Cruz
Post-docs:	Dr. Chloe Funkhouser (full) Dr. Ivan Guerrero Garcia (partial) Dr. Vikram Jadhao (full) Dr. Azita Parsaeian (full) Dr. Baofu Qiao (full) Dr. Rastko Sknepnek (full), Research Associate Dr. Sumanth Swaminathan (full) Dr. Kuo-An Wu (full) Dr. Creighton Thomas (full) Dr. Rui Zhang (full) Dr. Jos Zwanikken (partial) Dr. Zhenwei Yao, (partial) Dr. Aykut Erbas, (partial) Dr. Jiaye Su, (partial) Dr. Charles Sing (partial: International Institute for Nanotechnology Fellowship)
Graduate Students:	Ms. Ting Li (partial); Ph.D. degree in May 2015. Mr. Matthew Demers (full); Ph.D. degree in Aug 2010. Mr. Yufei Jing (partial) Mr. Shuangping Li (partial) Mr. Honghao Li (partial) Mr. Dongxu Huang (partial) Mr. Martin Girard (partial)
Visiting Scientists:	Prof. Vladimirs Kuzovkovs (Institute of Solid State Physics, Latvia University, Latvia) Prof. Pedro Gonzales-Mozuelos (Cinvestav del IPN, Mexico)

Thesis published as a result of research efforts:

1. Li, T. (2015). *DNA-programmable nanoparticle assembly and crystallization via multi-scale modeling and simulation* (Order No. 3724299). Available from ProQuest Dissertations & Theses Global. (1721417356). Retrieved from http://search.proquest.com/docview/1721417356?accountid=12861

Abstract

A challenge facing modern materials science research is how to design and create novel assemblies by controlling properties of nanoscale building blocks. Larger by over two orders of magnitude than conventional atoms, nanoparticles create materials possessing unique properties and offering broad applications in plasmonics, catalysis, photonics, etc. Nucleic-acid coated nanoparticles, which emerged in the mid-1990s, provide a unique platform in

which components such as nanoparticle size, shape, and composition along with DNA length, sequence, and coating density can all be modulated at will, thus allowing the identity of the building block and its bonding behavior to be independently tuned. In the following decades, the focus in this field has shifted from assembling amorphous materials to constructing basic crystalline structures, and subsequently to versatile one-, two-, and three-dimensional assemblies of anisotropic nanoparticles. Nowadays, defect-free single crystals have been realized, providing a promising feedstock for optically/mechanically functional devices. 4 To better design and predict novel assemblies, we apply molecular dynamics simulations alongside experiments to investigate the microscopic mechanism and identify key ingredients to a successful crystallization. A scale-accurate coarse-grained model with explicit DNA chains faithfully captures the relevant contributions to the kinetics of the DNA hybridization process. The model proves robust such that completely random initial configurations can self-assemble to recover all experimentally reported binary superlattices including body-centered cubic, CsCl, AlB2, Cr3Si and Cs6C60 symmetries. Based on this simulation approach, we map phase diagrams for different crystal symmetries, and propose suitable DNA linker sequences for future nanomaterials design. Faceted, single crystalline microcrystals consisting of DNA-functionalized nanospheres or socalled ‘programmable atom equivalents’ are synthesized using a slow-cooling approach. We show that nanoparticle crystals assembled in this manner form the Wulff equilibrium crystal structure expected from multi-scale modeling and simulations. Using the explicit-DNA model we calculate surface energy values for this system to predict the equilibrium polyhedra. Using a colloidal model with implicit DNA we study the crystal growth and rationalize the observed crystal shape. Despite operating at a much larger length scale and using fundamentally different interactions to drive assembly, the observed shape of the nanoparticle crystals is identical to the expected equilibrium crystal structure of a body-centered cubic packing of atoms. Thus, this finding establishes that DNA hybridization can direct nanoparticle assembly along a pathway that mimics atomic crystallization. We further study the effect of highly active hybridization on thermal fluctuations at surfaces. A mathematical relationship bridging controllable design parameters and surface energy fluctuations has been derived and validated via simulations. Based on this relationship, we suggest optimal design parameters favoring future design of stable single crystal polyhedra.

2. Demers, M. F. (2012). *Curvature-driven pattern formation in multicomponent membranes* (Order No. 3547829). Available from ProQuest Dissertations & Theses Global. (1269158445). Retrieved from http://search.proquest.com/docview/1269158445?accountid=12861

Abstract

Multicomponent liquid membranes occur in many important physical systems and offer promise in the rational design of materials. This promise is due in large part to their versatile phase behavior. Under the right circumstances, distinct physical properties of the components can lead to the spontaneous ordering of membrane components into patterns. This thesis contains a study of multicomponent membranes whose patterns arise from component-specific shape preferences. These preferences couple the membrane’s extrinsic and intrinsic geometries, since the membrane’s degrees of freedom include both its shape in space and its

in-membrane component arrangement. A continuum theory is developed for a multicomponent membrane with shape-composition coupling. A general scheme is presented for classifying strongly segregated membrane patterns according to the topology of component domains. This model is studied numerically and analytically. In the numerical study, a discretized model is formulated for the case of a strongly segregated three-phase vesicle. Minimal energy configurations are found using simulated 4 annealing Monte Carlo simulations. Phase diagrams are constructed for a number of control parameter pairs, exhibiting a wide variety of patterns. An effective attraction between components of largest and smallest spontaneous curvature is observed and characterized. In the analytic study, solutions for minimal surface shape in known patterns are computed for select tractable cases. These solutions are applicable to an arbitrary number of phase types, and applied to the case of a three-phase membrane.

Interactions/Transitions (partially or fully funded by this project, 2010-2015):

1. M. Olvera de la Cruz “DNA-functionalized nanoparticle assembly” ACS National Meeting, Denver, CO, March 22-27, 2015.
2. M. Olvera de la Cruz “DNA-functionalized Nanoparticle Assembly and Crystallization” SIAM Conference on Computational Science and Engineering, Salt Lake City, UT, March 14-18, 2015.
3. M. Olvera de la Cruz “Electrostatic Self-Assembly of Biomolecules,” APS Spring Meeting, San Antonio, TX, March 2-6, 2015.
4. Niels Boon, M. Olvera de la Cruz, “`Soft' amplifier circuits based on field-effect ionic transistors,” APS Spring Meeting, San Antonio, TX, March 2-6, 2015.
5. J. Zwanikken, M. Olvera de la Cruz “Tuning the phase diagram of polyelectrolyte blends with a pinch of salt,” APS Spring Meeting, San Antonio, TX, March 2-6, 2015.
6. A. Erbas, J. Zwanikken and M. Olvera de la Cruz “Electrostatics effects on normal load capacity of two like-charge hydrogels” APS Spring Meeting, San Antonio, TX, March 2-6, 2015.
7. T. Li, M. Olvera de la Cruz “DNA-programmable Nanoparticle Self-Assembly and Crystallization via Multi-Scale Modelling \& Simulation” APS Spring Meeting, San Antonio, TX, March 2-6, 2015.
8. S. Li, Z. Yao and M. Olvera de la Cruz “Perversions driven spontaneous symmetry breaking in heterogeneous elastic ribbons” APS Spring Meeting, San Antonio, TX, March 2-6, 2015.
9. Y. Jing, V. Jadhao, J. Zwanikken and M. Olvera de la Cruz “Electrostatic effects of dielectric interfaces on confined electrolyte” APS Spring Meeting, San Antonio, TX, March 2-6, 2015.
10. Z. Yao, M. Olvera de la Cruz “Dynamics of vacancies in two-dimensional Lennard-Jones crystals” APS Spring Meeting, San Antonio, TX, March 2-6, 2015.
11. S. Pan, T. Li and M. Olvera de la Cruz “Simulation of Epitaxial Growth of DNA-nanoparticle Superlattices on Pre-patterned Substrates” APS Spring Meeting, San Antonio, TX, March 2-6, 2015.

12. F.J. Solis, V. Jadhao, K. Mitra and M. Olvera de la Cruz “A variational free-energy functional approach to the Schrodinger-Poisson theory” APS Spring Meeting, San Antonio, TX, March 2-6, 2015.
13. M. Olvera de la Cruz “Electrostatic Self-Assembly of Biomolecules”, Advanced Workshop on Out-of-Equilibrium Matter, San Luis Potosi, Mexico, December 8-12, 2014.
14. M. Olvera de la Cruz, “DNA-Functionalized Nanoparticle Assembly and Crystallization,” Dept. of Materials Science and Eng. Colloquium, University of Illinois Urbana, November 17, 2014.
15. *M. Olvera de la Cruz, “Polyhedral Crystalline Membrane”, Department of Chem. Eng. Colloquium, Stanford University, October 27, 2014.
16. *M. Olvera de la Cruz, “Electrostatic Self-assembly of Biomolecules”, First “self-assembly of biomolecules” International Symposium, Montpellier, October 12-14, 2014.
17. *M. Olvera de la Cruz, “Ionic Membranes”, Department of Mechanical Eng. Colloquium, University of Illinois Urbana, September 2, 2014.
18. *M. Olvera de la Cruz, “Ionic Bilayers, Tail Packing and Mesoscale Geometry”, Workshop on Coarse-Grained Modeling of Polymers and Soft Materials for Genome Initiative, National Institute of Standards and Technology, August 6-7, 2014.
19. *M. Olvera de la Cruz, “Polyhedral Crystalline Membranes”, APS Colloquium, Argonne National Laboratory, Chicago, July 16, 2014.
20. *M. Olvera de la Cruz, “Crystalline Membranes’, Frontiers in Materials Sciences Seminar Series, Pacific Northwest National Laboratory, Richland, Washington, June 2, 2014.
21. *M. Olvera De La Cruz, “Electrostatic Driven Assembly”, Grand Challenges in Soft Matter Workshop, University of California, Santa Barbara, May 17-18, 2014
22. *M. Olvera de la Cruz, “Ion adsorption at solid-electrolyte interfaces” Lorentz Center, Leiden (NL) March 10-14, 2014.
23. *M. Olvera de la Cruz, Condensed Matter Seminar, James Franck Institute, Departments of Chemistry and Physics & Astronomy, University of Chicago, Feb.18, 2014.
24. *M. Olvera de la Cruz, International Symposium on Polyelectrolytes, Ein Gedi, Israel, Jan.20-23, 2014.
25. * M. Olvera de la Cruz, “Blebbing of Nuclear Lamin Networks”, American Society for Cell Biology Annual Meeting, Dec 14, 2013, New Orleans, LA.
26. *M. Olvera de la Cruz, “Molecular Crystallization and Mesoscale Geometry of Multicomponent Ionic Membranes”, Materials Research Society Fall Meeting, Dec 1-6, 2013, Boston, MA.
27. Charles Sing, Jos Zwanikken, and Monica Olvera de la Cruz "Effects of charge correlation on block copolyelectrolytes", AICHE National Meeting, Nov. 4, 2013, San Francisco, California (poster).
28. Charles Sing, Jos Zwanikken, and Monica Olvera de la Cruz "Ion correlations in block copolyelectrolytes", MRS National Meeting, Dec. 4, 2013, Boston, Massachusetts.
29. C. Sing and M. Olvera de la Cruz, "Effects of Ion Correlations on the Thermodynamics of Polymer Interfaces" ACS Fall Meeting (Indianapolis IN), Sept. 12 2013.
30. * M. Olvera de la Cruz, “Platonic and Archimedean geometries in elastic membranes” The 13th International conference on Properties and Phase Equilibria for Product and Process Design (PPEPPD), May 26-30, Iguaza Falls, Argentina.

31. *M. Olvera de la Cruz, “The Role of Electrolytes in the Assembly of Colloids” Gordon Research Conference: Self-Assembly & Supramolecular Chemistry, Les Diablerets, May 5-10, 2013, Switzerland
32. Charles Sing, Jos Zwanikken, and Monica Olvera de la Cruz "Ion-ion correlations in polymer systems" Gordon Research Seminar/Conference, May 5-10, 2013, Les Diablarets, Switzerland (poster).
33. * M. Olvera de la Cruz, “Charge and Composition Patterns in Ionic Fibers, Gels and Membranes” High Polymer Research Group Conference, April 28-May 3, 2013, UK.
34. * Charles E Sing, Jos Zwanikken, and Monica Olvera de la Cruz, "Highly-correlated charges in polyelectrolyte gels: Reentrant swelling and ion-ion correlations" in Celebrating 50 Years of Polymers at Case Western, ACS National Meeting & Exposition, April 9, 2013, New Orleans, Louisiana.
35. Guillermo I Guerrero-Garcia, and Monica Olvera de la Cruz "Inversion of the electric field at the electrified liquid-liquid interface”, ACS National Meeting & Exposition, April 7-11, 2013, New Orleans, Louisiana.
36. Charles Sing, Jos Zwanikken, and Monica Olvera de la Cruz "Highly-correlated charges in polymer systems: ion size and cohesion effects", ACS National Meeting & Exposition, April 7-11, 2013, New Orleans, Louisiana.
37. Baofu Qiao and Monica Olvera de la Cruz "Crystallization of ionic lipid membranes, elucidated by atomistic simulation”, ACS National Meeting & Exposition, April 7-11, 2013, New Orleans, Louisiana.
38. R. Zhang, P. Jha, and M. Olvera de la Cruz, “Non-equilibrium Ionic Assemblies of Oppositely Charged Colloids,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
39. Y. Jing, G.I. Guerrero Garcia, and M. Olvera de la Cruz, “Enhancing and reversing the electric field at liquid/liquid interfaces,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
40. F. Solis, V. Jadhao, and M. Olvera de la Cruz, “A variational formulation of electrostatics for heterogeneous dielectric media,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
41. B. Qiao, R.J. Ellis, and M. Olvera de la Cruz, “For a Safe Diamide Extraction Process, Elucidated by Atomistic Simulations,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
42. C.K. Thomas, and M. Olvera de la Cruz, “Why square lattices are not seen on curved ionic membranes,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
43. G.I. Guerrero Garcia, P. Gonzalez-Mozuelos, and M. Olvera de la Cruz, “Colloidal stability in concentrated electrolyte solutions using large counterions,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
44. K.L. Kohlstedt, M. Olvera de la Cruz, and G.C. Schatz, “Controlling orientational order of multivalent prisms in superlattice assemblies,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
45. A. Osorio-Vivanco, M. Olvera de la Cruz, and S. Glotzer, “Optimized assembly and steady-state length-scale control in dissipative systems of photo-switchable colloids,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.

46. T.I.N.G. Li, R. Sknepnek, and M. Olvera de la Cruz, “Hybridization dynamics to DNA guided crystallization,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
47. S. Kewalramani, C-Y. Leung, J. Zwanikken, R. Macfarlane, M. Olvera de la Cruz, C. Mirkin, and M. Bedzyk, “Determination of counterion distribution around DNA coated nanoparticles (DNA-AuNP) by small angle X-ray scattering (SAXS),” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
48. J. Zwanikken, and M. Olvera de la Cruz, “Tunable Soft Structure in Charged Fluids confined by Dielectric Interfaces,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
49. Z. Yao, and M. Olvera de la Cruz, “Packing of charged chains on toroidal geometries,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
50. C. Sing, J. Zwanikken, and M. Olvera de la Cruz, “Highly-correlated charges in polyelectrolyte gels,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
51. V. Jadhao, F. Solis, and M. Olvera de la Cruz, “Ion distributions near dielectric interfaces from Car-Parrinello molecular dynamics,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
52. C. Funkhouser, R. Sknepnek, T. Shimi, A. Goldman, R. Goldman, and M. Olvera de la Cruz, “An Elastic Model of Blebbing in Nuclear Lamin Meshworks,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
53. *M. Olvera de la Cruz, “Charge and Composition Patterns in Ionic Fibers, Gels and Membranes” High Polymer Research Group Conference, April 28-May 3, 2013, UK.
54. *M. Olvera de la Cruz, “Design of superlattices of Nucleic Acid-Gold Nanoparticles via DNA recognition” Brookhaven National Laboratory, Upton NY, April 9, 2013.
55. Charles E Sing, Jos Zwanikken, and Monica Olvera de la Cruz, "Highly-correlated charges in polyelectrolyte gels: Reentrant swelling and ion-ion correlations" in Celebrating 50 Years of Polymers at Case Western, ACS National Meeting & Exposition, April 7-11, 2013, New Orleans, Louisiana.
56. Guillermo I Guerrero-Garcia, and Monica Olvera de la Cruz "Inversion of the electric field at the electrified liquid-liquid interface”, ACS National Meeting & Exposition, April 7-11, 2013, New Orleans, Louisiana.
57. Baofu Qiao and Monica Olvera de la Cruz "Crystallization of ionic lipid membranes, elucidated by atomistic simulation”, ACS National Meeting & Exposition, April 7-11, 2013, New Orleans, Louisiana.
58. *M. Olvera de la Cruz, “Polyhedral Geometries in the Living World,” Dept. of Chemical Engineering, University of Texas at Austin, Austin TX, April 4, 2013.
59. R. Zhang, P. Jha, and M. Olvera de la Cruz, “Non-equilibrium Ionic Assemblies of Oppositely Charged Colloids,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
60. Y. Jing, G.I. Guerrero Garcia, and M. Olvera de la Cruz, “Enhancing and reversing the electric field at liquid/liquid interfaces,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
61. F. Solis, V. Jadhao, and M. Olvera de la Cruz, “A variational formulation of electrostatics for heterogeneous dielectric media,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.

62. B. Qiao, R.J. Ellis, and M. Olvera de la Cruz, “For a Safe Diamide Extraction Process, Elucidated by Atomistic Simulations,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
63. C.K. Thomas, and M. Olvera de la Cruz, “Why square lattices are not seen on curved ionic membranes,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
64. C-Y. Leung, M. Greenfield, S. Kewalramani, L. Palmer, R. Sknepnek, B. Qiao, C. Newcomb, G. Vernizzi, M. Bedzyk, S. Stupp, and M. Olvera de la Cruz, “Mesoscopic Membrane Morphology Regulated by Molecular Crystallization,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
65. G.I. Guerrero Garcia, P. Gonzalez-Mozuelos, and M. Olvera de la Cruz, “Colloidal stability in concentrated electrolyte solutions using large counterions,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
66. K.L. Kohlstedt, M. Olvera de la Cruz, and G.C. Schatz, “Controlling orientational order of multivalent prisms in superlattice assemblies,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
67. A. Osorio-Vivanco, M. Olvera de la Cruz, and S. Glotzer, “Optimized assembly and steady-state length-scale control in dissipative systems of photo-switchable colloids,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
68. T.I.N.G. Li, R. Sknepnek, and M. Olvera de la Cruz, “Hybridization dynamics to DNA guided crystallization,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
69. S. Kewalramani, C-Y. Leung, J. Zwanikken, R. Macfarlane, M. Olvera de la Cruz, C. Mirkin, and M. Bedzyk, “Determination of counterion distribution around DNA coated nanoparticles (DNA-AuNP) by small angle X-ray scattering (SAXS),” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
70. J. Zwanikken, and M. Olvera de la Cruz, “Tunable Soft Structure in Charged Fluids confined by Dielectric Interfaces,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
71. Z. Yao, and M. Olvera de la Cruz, “Packing of charged chains on toroidal geometries,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
72. C. Sing, J. Zwanikken, and M. Olvera de la Cruz, “Highly-correlated charges in polyelectrolyte gels,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
73. V. Jadhao, F. Solis, and M. Olvera de la Cruz, “Ion distributions near dielectric interfaces from Car-Parrinello molecular dynamics,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
74. C. Funkhouser, R. Sknepnek, T. Shimi, A. Goldman, R. Goldman, and M. Olvera de la Cruz, “An Elastic Model of Blebbing in Nuclear Lamin Meshworks,” American Physical Society March Meeting, Baltimore, MD, March 18-22, 2013.
75. *M. Olvera de la Cruz, “Polyhedral geometries in crystalline membranes”, XLII Winter meeting on Statistical Physics, January 8-11, 2012, Taxco, Mexico.
76. *M. Olvera de la Cruz, “Platonic and Archimedean Geometries in Elastic Membranes”, Materials Research Society Fall Meeting, Nov 27, 2012, Boston, MA.
77. S. Patala, L. Marks, and M. Olvera de la Cruz, “Stability Analysis for Faceted Pentagonal Nanoparticles”, Materials Research Society Fall Meeting, Nov 30, 2012, Boston, MA.

78. *M. Olvera de la Cruz, “Modeling mesoscale phenomena in crystalline membranes”, SACNAS National Conference, October 11-14, 2012, Seattle WA.
79. *M. Olvera de la Cruz, “Polyhedral Geometries in the Living World”, Condensed Matter Seminar, Dept. of Physics, University of Illinois at Urbana-Champaign, September 21, 2012, Urbana IL.
80. *M. Olvera de la Cruz, “Computational efforts in Polymer Science”, 244th ACS National Meeting, August 19-23, 2012, Philadelphia, PA.
81. *M. Olvera de la Cruz, “Computational modeling of polyelectrolyte gels: Charge regulation and nanoscale phase behavior”, 244th ACS National Meeting, August 19-23, 2012, Philadelphia, PA.
82. *M. Olvera de la Cruz, “Platonic and Archimedean Geometries in Multicomponent Elastic Membranes” XXI International Materials Research Congress (IMRC), August 13-17, 2012, Cancun, Mexico.
83. *M. Olvera de la Cruz, “The stability of polyvalent nanoparticles and effective interactions in molecular electrolytes” XXI International Materials Research Congress (IMRC), August 13-17, 2012, Cancun, Mexico.
84. *M. Olvera de la Cruz, “Surprises in ionic driven assembly of membranes” Argonne National Laboratory, August 7, 2012, Argonne, IL.
85. *M. Olvera de la Cruz, “Charge and Composition Patterns in Ionic Membranes” Recent Progresses on Coulomb Many-body Systems Workshop, Shanghai Jiao Tong University, June 9-16, 2012, Shanghai, China.
86. *M. Olvera de la Cruz, “Polyelectrolyte Gels” International Symposium on Polymer Physics, June 4-8, 2012, Chengdu, China.
87. *M. Olvera de la Cruz, “Ionic driven assembly of membranes: Surprising findings in shell shape and composition”, May 3, 2012, Dept. of Physics, North Dakota State University.
88. *M. Olvera de la Cruz, “Physical Properties of Heterogeneous Microcompartments”, Procter and Gamble Lecture Series, Spring 2012, Department of Chemistry and Biochemistry, University of California, April 16, 2012, Los Angeles, CA.
89. *M. Olvera de la Cruz, “Modeling heterogeneous fibers and membranes” (in "Computational Materials Design In Heterogeneous Systems") MRS Spring meeting, April 9-13, 2012, San Francisco, CA.
90. C. Thomas and M. Olvera de la Cruz, “Charge correlations in multicomponent ionic crystalline membranes”, American Physical Society March Meeting, Boston, MA, February 27- March 2, 2012.
91. M. Olvera de la Cruz, J. Zwanikken and C.A. Mirkin, “Locial ionic environment around polyvalent nucleic-acid functionalized gold nanoparticles”, American Physical Society March Meeting, Boston, MA, February 27- March 2, 2012.
92. M. Demers, R. Sknepnek and M. Olvera de la Cruz, “Curvature driven domain formation in ternary lipid membranes”, American Physical Society March Meeting, Boston, MA, February 27- March 2, 2012.
93. R. Sknepnek and M. Olvera de la Cruz, “Thin-shell model for faceting of multicomponent elastic vesicles”, American Physical Society March Meeting, Boston, MA, February 27- March 2, 2012.

94. T. Li, R. Sknepnek, R.J. Macfarlane, C.A. Mirkin and M. Olvera de la Cruz, “Modeling of DNA-directed colloidal self-assembly and crystallization”, American Physical Society March Meeting, Boston, MA, February 27- March 2, 2012.
95. C-Y. Leung, L. Palmer, S. Kewalramani, R. Sknepnek, G. Vernizzi, M. Greenfield, S. Stupp, M. Bedzyk and M. Olvera de la Cruz, “Electrostatics-driven assembly of uni-lamellar cationic facetted vesicles”, American Physical Society March Meeting, Boston, MA, February 27- March 2, 2012.
96. J. Zwanikken and M. Olvera de la Cruz, “Ion-induced interactions between charged macroions and dielectric inhomogeneities”, American Physical Society March Meeting, Boston, MA, February 27- March 2, 2012.
97. P. Jha, J. Zwanikken, F. Detcheverrry, J. de Pablo and M. Olvera de la Cruz, “Influence of charge and network inhomogeneities on the swollen-collapsed transition in polyelectrolyte nanogels”, American Physical Society March Meeting, Boston, MA, February 27- March 2, 2012.
98. J. Su, H. Guo and M. Olvera de la Cruz, “Solubility and transport of cationic and anionic patterned nanoparticles”, American Physical Society March Meeting, Boston, MA, February 27- March 2, 2012.
99. B. Grzybowski, M. Olvera de la Cruz, P. Jha and V. Kuzovkov, “A novel kinetic Monte Carlo algorithm for non-equilibrium simulations”, American Physical Society March Meeting, Boston, MA, February 27- March 2, 2012.
100. G.I. Guerrero Garcia and M. Olvera de la Cruz, “Ion correlations in the electrical double layer near liquid/liquid interfaces”, American Physical Society March Meeting, Boston, MA, February 27- March 2, 2012.
101. C. Funkhouser, R. Sknepnek and M. Olvera de la Cruz, “Morphologies of elastic membranes with fluctuating connectivity”, American Physical Society March Meeting, Boston, MA, February 27- March 2, 2012.
102. B. Qiao and M. Olvera de la Cruz, “Effect of valence of counterions on the structure of charged membranes, a computer simulation study”, American Physical Society March Meeting, Boston, MA, February 27- March 2, 2012.
103. *M. Olvera de la Cruz, “Surpises in ionic driven assembly of membranes”, Caltech Chemical Physics Seminar Series, California Institute of Technology, January 31, 2012, Pasadena, CA.
104. *M. Olvera de la Cruz, “Surprises in electrostatic driven assembly of ionic crystalline shells”, Department of Materials Science, University of Michigan, November 18, 2011, Ann Arbor, MI.
105. *M. Olvera de la Cruz, J. W. Zwanikken and C. A. Mirkin “Ionic screening and ion-induced attractions in solutions of nanoparticles” First Workshop on Advances in Colloidal Materials- 25th Anniversary-Biocolloid and Fluid Physics Group (1986-2011), University of Granada, September 23, 2011, Granada, Spain.
106. *M. Olvera de la Cruz, “Ionic Crystalline Shells”, Colloquium, Dept. of Chemical & Biological Engineering, Rensselaer Polytechnic Institute, September 14, 2011, Troy, NY.
107. *M. Olvera de la Cruz, “Platonic and Archimedean Geometries in Elastic Membranes”, Colloquium, Applied Physics, Harvard University, September 9, 2011, Cambridge, MA.

108. *M. Olvera de la Cruz, “New Geometries of Elastic Closed Membranes and Crystalline Shells”, Laboratoire de Physique de Solides, University of Orsay, July 12, 2011, Orsay, France.
109. *M. Olvera de la Cruz, “Heterogeneous Elastic Membranes: New Shapes of Microcompartments”, Telluride Workshop on Polymer Physics, June 20-24, 2011, Telluride, CO.
110. *M. Olvera de la Cruz, J. W. Zwanikken, P. Guo, R. J. Macfarlane, and C. A. Mirkin, “Grafting density effect on ionic screening around functionalized Nanoparticles”, 241st ACS National Meeting & Exposition - March 27-31, 2011, Anaheim, California.
111. G. Ivan Guerrero-Garcıa, P. Gonzalez-Mozuelos, and M. Olvera de la Cruz, “On the interaction of equally charged nanoparticles in presence of a size-asymmetric salt”, 241st ACS National Meeting & Exposition - March 27-31, 2011, Anaheim, California.
112. J. W. Zwanikken and Monica Olvera de la Cruz, “Ions and charged macromolecules near the interface between two electrolyte solutions”, 241st ACS National Meeting & Exposition - March 27-31, 2011, Anaheim, California.
113. *M. Olvera de la Cruz, “Responsive Polyelectrolyte Gels and Tethered Membranes”, American Physical Society March Meeting, Dallas, TX, March 21-25, 2011.
114. F. J. Solis and M. Olvera de la Cruz, “Ionic conduction at liquid-liquid interfaces”, American Physical Society March Meeting, Dallas, TX, March 21-25, 2011.
115. P. Jha, J. Zwanikken, F. Detcheverry, J. de Pablo, M. Olvera de la Cruz, “Theoretically informed coarse-grained simulations of polymer nanogels”, American Physical Society March Meeting, Dallas, TX, March 21-25, 2011.
116. P. Guo, R. Sknepnek, M. Olvera de la Cruz, “Ridge formation of charged end group ligands grafted on faceted nanoparticle”, American Physical Society March Meeting, Dallas, TX, March 21-25, 2011.
117. V. Jadhao, F. J. Solis, G. Guerrero-Garcia, M. Olvera de la Cruz, “Towards simulation of charges in the presence of varying dielectric response”, American Physical Society March Meeting, Dallas, TX, March 21-25, 2011.
118. M. Demers, F. J. Solis, M. Olvera de la Cruz, “Pattern formation in ternary lipid membranes with composition- deformation coupling”, American Physical Society March Meeting, Dallas, TX, March 21-25, 2011.
119. S. Dhakal, F. J. Solis, M. Olvera de la Cruz, “Orientational order and defect structures on curved surfaces”, American Physical Society March Meeting, Dallas, TX, March 21-25, 2011.
120. J. Zwanikken, M. Olvera de la Cruz, “Correlated electrolyte solutions and ion-induced attractions between nanoparticles”, American Physical Society March Meeting, Dallas, TX, March 21-25, 2011.
121. C. Y. Leung, R. Sknepnek, L. Palmer, G. Vernizzi, M. Greenfield, S. Stupp, M. Bedzyk, M. Olvera de la Cruz, “Crystallization induced by electrostatic correlations in vesicles of mixed-valence ionic amphilphiles”, American Physical Society March Meeting, Dallas, TX, March 21-25, 2011.
122. S. Swaminathan, F. J. Solis, M. Olvera de la Cruz, “Conformation and mechanical properties of diblock fibers”, American Physical Society March Meeting, Dallas, TX, March 21-25, 2011.
123. R. Sknepnek, C. Leung, L. C. Palmer, G. Vernizzi, S. I. Stupp, M. J. Bedzyk, M. Olvera de la Cruz, “Faceting of multicomponent charged elastic shells”, American

Physical Society March Meeting, Dallas, TX, March 21-25, 2011.

124. M. Olvera de la Cruz

125. 14A. Parsaeian, J. F. Marko, M. Olvera de la Cruz, " Binding-rebinding dynamics of proteins interacting non- specfically with a long DNA molecule", American Physical Society March Meeting, Dallas, TX, March 21-25, 2011.

126. *M. Olvera de la Cruz "Regular and Irregular Polyhedra in Multi-Component Crystalline Shells", Fay Ajzenberg-Selove Colloquium, Physics Department, University of Wisconsin, Madison, Feb. 18, 2011.

127. M. Olvera de la Cruz "Regular and Irregular Polyhedra in Multi-Component Elastic Membranes" Workshop on Self-Assembled Bio-Inspired Materials for Energy, Argonne, February 4, 2011.

128. *M. Olvera de la Cruz "Heterogeneous Membranes" Colloquium, Department of Chemical Engineering, University of Illinois at Chicago, January 13, 2011.

129. M. Olvera de la Cruz, "Surprises in Heterogeneous Elastic Membranes" Colloquium, Department of Polymer Science and Eng., University of Massachusetts, Amherst Physics, September 24, 2010.

130. M. Olvera de la Cruz, "Ionic Membranes and Gels" **Plenary Lecture**, 2nd International Soft Matter Conference (ISMC 2010), Granada Spain July 5-8, 2010.

131. M. Olvera de la Cruz, "Heterogeneous Elastic Membranes" Self-assembly in Biology and Materials Science Workshop, Huatulco, Oaxaca, June 9-11, 2010.

132. M. Olvera de la Cruz, "Symmetries Broken by Electrostatics in Nanoscale Ionic Assemblies", **Plenary Speaker**, Society of Industrial and Applied Mathematics (SIAM) meeting on Mathematical Aspects of Materials Science, Philadelphia, PA, May 23-26, 2010.

133. M. Olvera de la Cruz, "Self-Assembly in Molecular Electrolytes" **Plenary Talk**, The 4th PENN-UPRH PREM Symposium on Soft Matters in Materials Science, Humacao, Puerto Rico, May 7, 2010.

(***** invited presentations)

Awards and Honors:

Prof. Monica Olvera de la Cruz

- 2013-2015 Basic Energy Sciences Advisory Committee, Department of Energy
- 2014 National Science Foundation Advisory Committee for International Science and Engineering
- 2013 Derieux Lecture, Department of Physics, North Carolina State University
- 2013 Distinguished Lecturer, Mathematical Physical Sciences Directorate, National Science Foundation
- 2012 Elected Member of the National Academy of Sciences
- 2011 Fay Ajzenberg-Selove Colloquium Speaker, Physics Department, University of Wisconsin, Madison
- 2010 Special Civil Merit Award, State of Guerrero, Mexico
- 2010 American Academy of Arts and Sciences Fellow
- 2010-2012 Chair, Condensed Matter and Materials Research Committee, National Research Council, the National Academy of Sciences.

- 2009-2015 Board of Physics and Astronomy, National Research Council, the National Academy of Sciences (membership was renewed in 2011-13 and 2013-15)

Dr. Chloe Funkhouser

June 2012 *Most Novel Project* prize for poster entitled "*Modeling and Simulations of the Nuclear Lamina: Nuclear Blebbing*" at the12th biennial Gordon Conference on Intermediate Filaments.

Dr. Charles Sing

2012-2014 Inaugural *International Institute of Nanotechnology Postdoctoral Fellowship*. This fellowship provides 50% of the funding for Dr. Sing, with the other half funded by PI's NSSEFF award.

Change in AFOSR program manager, if any:

In year 1, the program manager was Dr. Djuana Lea. The program manager thereafter was Dr. Julie Moses.

Program Codes

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <iostream>
#include <time.h>
#include <iomanip>
#include <fstream>
using namespace std;
#include "utils/nr.h"
#include "gnuplot_i.hpp"
#include <gsl/gsl_dht.h>
#include <gsl/gsl_sf_bessel.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_rng.h>

#if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) || defined(__TOS_WIN__)
 #include <conio.h>   //for getch(), needed in wait_for_key()
 #include <windows.h> //for Sleep()
#endif

void wait_for_key(); // Program halts until keypress

#define pi 3.14159265358979238462643
#define NA 0.60221415
#define e0 8.85418781762O-12
#define e 1.60217646e-19
#define kB 1.3806503e-23

#define T 293.0
#define ew 80.1 /* at 293 K */ //78.4

#define Large 1e10
/* ----------------------------------------------------------------------------------------------
    CoulombFluid solver

    Quantitative method predicting the mean density profiles and correlation functions
    in inhomogeneous Coulomb fluids. The 3D inhomogeneous system is mapped on a multicomponent
    2D homogeneous system, which enables the application of Ornstein-Zernike theory.

    December 13 2011
    Jos Zwanikken
    Monica Olvera de la Cruz Group
    Materials Science and Engineering
    2220 Cook Hall
    Northwestern University

    Please cite my latest papers, or at least a few related to the subject such as

    [1] Tunable soft structure in charged fluids confined by dielectric interfaces
    J. W. Zwanikken and M. Olvera de la Cruz, Proc. Natl. Acad. Sci. 110 (14) (2013),  5301â€“5308

    [2] Moderately coupled charged fluids near dielectric interfaces and in confinement
    J. W. Zwanikken, Electrostatics of Soft and Disordered Matter, D. S. Dean, J. Dobnikar, A. Naji, and R. Podgornik Eds., Proceedings of the CECAM Workshop "New challenges in Electrostatics of Soft and Disordered Matter" (Pan Stanford, 2013 in press)

    [3] Effect of ion-ion correlations on Polyelectrolyte Gel Collapse and Reentrant Swelling
    Charles E. Sing, Jos W. Zwanikken, and Monica Olvera de la Cruz, Macromolecules (2013), accepted

    [4] Understanding Swollen-Collapsed and Re-entrant Transitions in Polyelectrolyte Nanogels by a Modified Donnan Theory
    Prateek K. Jha, J. W. Zwanikken, and M. Olvera de la Cruz, Soft Matter, 8, (37) (2012), 9519-9522


---------------------------------------------------------------------------------------------- */


class CoulombFluid {
    Mat3D_DP h, c, y, newh, hInit, hFin, newg, cInit, cFin, newc, u, uHT, uReg, uRegHT, Qs, QsHT, hFT;
    Mat_DP delta, htest, mu, newMu, Zcorr;
    Vec_DP n, nOld, z, dz, r, kFT;
    double cs1, cs2, lB1, lB2, kappa1, kappa2, R1, R2, ap, am, e1, e2, e3, H, H0, qp, qm, epsilon, phiD, muEx1p, muEx1m, muEx2p, muEx2m, temp, aReg, depth, BulkCutoff, Vtot,GridCutoff,Htot;
    double *dc,*tc,*dcHT,*tcHT, *cp, *cm, *QDNA, *phi, *Vextp, *Vextm, *cpMF, *cmMF, *phiMF, *cpAN, *cmAN, *phiAN, *muexp, *muexm, *cpInit, *cmInit, *cpFin, *cmFin;
    double *Fexc, *Fid, *Fext, *FexcDH, *Pzz, *Z1Z1, *Z2Z2, *Z1Z2, *NP, *NM, *qpv, *qmv, *apv, *amv, *Hf, *Theta, *ThetaGauss, cs1p, cs1m, cs2p, cs2m, Vlength, Lw, period;
    double errmax,errmax0,err,tolerance,alpha,alpha0[2], Fref, sigma, sigmaDNA;
    double  *errMem, errLimitSq, dummy, alphaMin;
    int Nr, Nz, Nb, Nf, Nl, nf, i, j, k, iterations, evaluations, interface, jmax, kmax, Nerr;
    bool correlations, convergence, Overrule, BulkPhase2, Singularities,AskQuestion;
    char project[70];
    Gnuplot g1,g2;

    void ResetValues();
    void SetGrid();
    void SetGrid2Walls();
    void SetQDNA(double);
    void SetGrid2Walls1Dielectric();
    void SetGridExcluded();
    void InformationFile(bool);
    void SetPairPotential();
    void SetPairPotential2Walls();
    void SetPairPotential2DielectricWalls();
    void SetRegularizedPairPotential2DielectricWalls();
    void SetPairPotential2Walls1Dielectric();
    void GnuplotDensity();
    void GnuplotCorrelationFunctions();
    void SetGrid2WallSession(double);
    void InitiateCorrelationFunctions(double*,double*,double*,double*,double*,double*,double*,int,int);
    void InitiateCorrelationFunctions2Walls(double*,double*,double*,double*,double*,double*,double*,int,int);
    void SetHankelTransformPairPotential();
    void EvaluateReservoirMuEx (int);
    void InitialCorrelationFunctions();
```

```
    void SetCorrelationFunctions();
    void FinalCorrelationFunctions();
    void GetCorrelationFunctions();
    void ResetQs();
    void OZ_HNC();
    void OZ_HNC_ShortRange();
    void OZ_HNC_Alternative();
    void AHNC_Loop(int*,bool,double);
    void AHNC_convolute_Loop(int*,bool,double);
    void AMHNC_Loop(int*,bool,double);
    void OZ_AHNC_AdaptiveFinder(double);
    void CalculateDensity(int);
    void CalculateMeanPotentialSlit();
    void CalculateMeanPotential1boundary();
    void CalculateExpensivePotential();
    void CalculatePotentialWithTraceSalt();
    void Potentiostat();
    void CalculateEfieldAndPotential();
    void CalculateExpensivePotential2Walls();
    void CalculateAnalyticalProfiles();
    void CalculateFreeEnergy();
    void CalculateCorrelationLength();
    void SymmetrizeFunctions(int);
    void RecordTemporaryProfiles();
    void RecordProfiles(bool);
    void RecordSnapShot();
    void RecordCorrelationFunctions();
    void RecordBulkCorrelationFunctions(double*,double*,double*,double*,int,int);
    void RecordTestfunctions();
    void RecordConvolutionFunctions(double*);
    void RecordTime(int*);
    void RecordFreeEnergy(bool);


  public:
    CoulombFluid();
    void TestFunction();
    void TestFunction2();
    void TestFunction3();
    void TestReservoir();
    void TestMatrixInversion();
    void TestConvolution();
    void TestGaussianConvolution();
    void test_dht_potentials();
    void SetName(char*);
    void Iterate();
    void Iterate2Walls();
    void Iterate2DielectricWalls();
    void Iterate2Walls1Dielectric();
    void Iterate2WallSession();
    void Iterate2DielectricWallSession();
    void ObtainDisjoiningPressure();
};
```

```
CoulombFluid::CoulombFluid() {

  g1.set_title("Density Profiles");
  g2.set_title("Anisotropic Correlation Function h_{+-}");

  qp = 2.0;//1.4; /* cation charge (in units of e) */
  qm = 1.0;//1.4; /* anion charge (in units of e) */
  cs1 = 2.0*50.0e-3 / qp;//2.5e-1;//1.66666667e-1; /* concentration cations in M in medium 1*/
  //  cs2 = 0.5e-3;  cs2 *= NA; /* concentration cations in M in medium 2*/
  cs1 *= NA; /* concentration in # per nm^2 */
  ap = 0.357;//57;//2125; /* cation radius in nm */
  am = 0.357;//57;//2125; /* anion radius in nm */
  e1 = 80.0;//ew;//2.0;//ew; /* relative permittivity in medium 1 */
  e2 = 80.0;//ew;//70;//34.8; /* relative permittivity in medium 2 */
  e3 = 80.0;//ew;//ew*ew/70;
  lB1 = e*e/(4*pi*e0*e1*kB*T) *1.0e9; /* Bjerrum length in medium 1 in nm*/
  lB2 = e*e/(4*pi*e0*e2*kB*T) *1.0e9; /* Bjerrum length in medium 2 in nm */
  kappa1 = sqrt(4*pi*lB1*(qp+qm)*qp*cs1); /* inverse Debye length in medium 1 in 1/nm */
  //  kappa2 = sqrt(8*pi*lB2*qp*cs2); /* inverse Debye length in medium 2 in 1/nm */
  2*ap*kappa1 + 4.0 < 20.0*kappa1 ? H = 20.0*kappa1 : H = 2*ap*kappa1 + 4.0; /* screening lengths from the interface */

  cs2 = qp/qm*cs1*exp(-0.25*(qp*qp/ap+qm*qm/am)*(lB2-lB1));
  cs2 = cs1;//2.7e-3*NA; /* Density of free BTPPA- TPFB+ */
  kappa2 = sqrt(8*pi*lB2*qp*cs2);
  phiD = 0.25*(qp*qp/ap-qm*qm/am)*(lB1-lB2);

  Vtot = 0.0; //0.05; //total electrostatic potential difference over the plates, in Volt
  sigma = 2*pi*lB1 *pow(1.0-(e1-e2)/(e1+e2)*(e1-e3)/(e1+e3),-1.0);
  sigma = Vtot / (6*pi*lB1) * e / (kB*T);
  sigma = 5*Vtot / (6*pi*lB1) * e / (kB*T);

  depth = 0.0; /* potential well at the surface in kT (positive value gives attractive wall) */

  aReg = 2*ap;

  ap + 8.0/kappa1 < 40.0 ? R1 = 40.0 : R1 = ap + 8.0/kappa1; /* cutoff distance from central ion in medium 1 */
  ap + 8.0/kappa2 < 40.0 ? R2 = 40.0 : R2 = ap + 8.0/kappa2; /* cutoff distance from central ion in medium 2 */

//kappa1=kappa2; //ALERt for 1Dielectric algorithm only

  H = 1*sqrt(e/1e-21)*kappa1; //6.0;
  //R1 = R2 = 5.0;//15.5/kappa1+2*ap;// 6*max(ap,am) + 6.0/kappa1;
  R1 = R2 = max(0.0,4.5/kappa1+2*ap);// 6*max(ap,am) + 6.0/kappa1;

  H = 2.0*kappa1;//+2*ap*kappa1;//*kappa1;//3.0*kappa1; //6.0;
//  R1 = R2 = ap + 10.0-ap;// /kappa2;
//  H = 2*ap*kappa1 + 3.0;
  Htot = 3.0;
  sigmaDNA = 0.0; //DNA coverage density in 1/nm^2

  tolerance = 1e-9;
```

```
alpha = 0.001;

Nerr = 10;
errMem = (double*) calloc (Nerr,sizeof(double));
errLimitSq = 1e-4*(Nerr-2);
alphaMin = 0.05; //0.05


period = ap*pi*pi*10;

evaluations = 0;

Nr = 50;
Nz = 50;
Nb = 1024/2;
Nf = 1*40; nf = 0;

h.assign(0.0,Nr,2*Nz,2*Nz);
c.assign(0.0,Nr,2*Nz,2*Nz);
y.assign(0.0,Nr,2*Nz,2*Nz);
u.assign(0.0,Nr,2*Nz,2*Nz);
uHT.assign(0.0,Nr,2*Nz,2*Nz);
uReg.assign(0.0,Nr,2*Nz,2*Nz);
uRegHT.assign(0.0,Nr,2*Nz,2*Nz);
newh.assign(0.0,Nr,2*Nz,2*Nz);
hInit.assign(0.0,Nr,2*Nz,2*Nz);
hFin.assign(0.0,Nr,2*Nz,2*Nz);
newg.assign(0.0,Nr,2*Nz,2*Nz);
cInit.assign(0.0,Nr,2*Nz,2*Nz);
cFin.assign(0.0,Nr,2*Nz,2*Nz);
newc.assign(0.0,Nr,2*Nz,2*Nz);
Qs.assign(0.0,Nr,2*Nz,2*Nz);
QsHT.assign(0.0,Nr,2*Nz,2*Nz);
hFT.assign(0.0,Nr,2*Nz,2*Nz);

delta.assign(0.0,2*Nz,2*Nz);
htest.assign(0.0,Nr,2*Nz);
mu.assign(0.0,2*Nz,2*Nz);
newMu.assign(0.0,2*Nz,2*Nz);
Zcorr.assign(0.0,2*Nz,2*Nz);
for (i=0;i<2*Nz;i++) delta[i][i] = 1.0;

n.assign(0.0,2*Nz);
nOld.assign(0.0,2*Nz);
z.assign(0.0,Nz);
dz.assign(0.0,Nz);
r.assign(0.0,Nr);
kFT.assign(0.0,Nr);

dc = (double*) calloc (Nr,sizeof(double));
tc = (double*) calloc (Nr,sizeof(double));
dcHT = (double*) calloc (Nr,sizeof(double));
tcHT = (double*) calloc (Nr,sizeof(double));
cp = (double*) calloc (Nz,sizeof(double));
cm = (double*) calloc (Nz,sizeof(double));
QDNA = (double*) calloc (Nz,sizeof(double));
phi = (double*) calloc (Nz,sizeof(double));
Vextp = (double*) calloc (Nz,sizeof(double));
Vextm = (double*) calloc (Nz,sizeof(double));
cpMF = (double*) calloc (Nz,sizeof(double));
cmMF = (double*) calloc (Nz,sizeof(double));
phiMF = (double*) calloc (Nz,sizeof(double));
cpAN = (double*) calloc (Nz,sizeof(double));
cmAN = (double*) calloc (Nz,sizeof(double));
cpInit = (double*) calloc (Nz,sizeof(double));
cmInit = (double*) calloc (Nz,sizeof(double));
cpFin = (double*) calloc (Nz,sizeof(double));
cmFin = (double*) calloc (Nz,sizeof(double));
phiAN = (double*) calloc (Nz,sizeof(double));
muexp = (double*) calloc (Nz,sizeof(double));
muexm = (double*) calloc (Nz,sizeof(double));
Hf = (double*) calloc (Nf,sizeof(double));
Fexc = (double*) calloc (Nf,sizeof(double));
Fid = (double*) calloc (Nf,sizeof(double));
Fext = (double*) calloc (Nf,sizeof(double));
FexcDH = (double*) calloc (Nf,sizeof(double));
Pzz = (double*) calloc (Nf,sizeof(double));
Z1Z1 = (double*) calloc (Nf,sizeof(double));
Z2Z2 = (double*) calloc (Nf,sizeof(double));
Z1Z2 = (double*) calloc (Nf,sizeof(double));
NP = (double*) calloc (Nf,sizeof(double));
NM = (double*) calloc (Nf,sizeof(double));

Theta = (double*) calloc (Nr,sizeof(double));
ThetaGauss = (double*) calloc (Nr,sizeof(double));

qpv = (double*) calloc (Nz,sizeof(double));
qmv = (double*) calloc (Nz,sizeof(double));
apv = (double*) calloc (Nz,sizeof(double));
amv = (double*) calloc (Nz,sizeof(double));

for (j=0;j<Nz;j++) { qpv[j]=qp; qmv[j]=qm; apv[j]=ap; amv[j]=am; }
//  for (j=0;j<Nz;j++) { if(j<Nz/2) {qpv[j]=1.0; qmv[j]=1.0; apv[j]=0.2125; amv[j]=0.2125;} else {qpv[j]=1.0; qmv[j]=1.0; apv[j]=0.425; amv[j]=0.425;} }

for (i=0;i<Nz;i++) { cp[i] = cs1; cm[i] = qpv[i]/qmv[i]*cs1; }

strcpy(project, "default");

Overrule = false;
BulkPhase2 = false;
Singularities = false;
AskQuestion = true; // check for not overwriting data. Value need not be changed
BulkCutoff = 1.0;//kappa2;
Lw=0.5;
Nl=0;
```

```
  GridCutoff=1.7; //0 cuts off the grid at a distance GridCutoff*ap/2.0 from the boundary

}

void CoulombFluid::ResetValues() {

  ap = 0.37;//57;//2125; /* cation radius in nm */
  am = 0.37;//57;//2125; /* anion radius in nm */

  e1=ew;
//  e1 = ew;
  cs1 = 4.0e-1 + 1.0e-1*temp; /* concentration cations in M in medium 1*/ cout << "\ncs1 = " << cs1 << endl;
  cs1 *= NA; cs2 = cs1;
  qp = 1.0 + 0.0*temp; /* cation charge (in units of e) */ cout << "qp = " << qp << endl;
  qm = 1.0 + 0.0*temp; /* anion charge (in units of e) */ cout << "qm = " << qm << "\n" << endl;
  for (j=0;j<Nz;j++) { qpv[j]=qp; qmv[j]=qm; apv[j]=ap; amv[j]=am; }
//  e2 = ew*temp;//34.8; /* relative permittivity in medium 2 */
  lB1 = e*e/(4*pi*e0*e1*kB*T) *1.0e9; /* Bjerrum length in medium 1 in nm*/
  lB2 = e*e/(4*pi*e0*e2*kB*T) *1.0e9; /* Bjerrum length in medium 2 in nm */
  kappa1 = sqrt(8*pi*lB1*qp*cs1); /* inverse Debye length in medium 1 in 1/nm */
  kappa2 = sqrt(8*pi*lB2*qp*cs2); /* inverse Debye length in medium 2 in 1/nm */
  kappa1=kappa2; //ALERT for 1Dielectric algorithm only
  R1 = R2 = max(0.0,3.5/kappa1+2*ap);// 6*max(ap,am) + 6.0/kappa1;
//  H = max(0.0*kappa1,0.0+1.5*ap*kappa1);//3.0*kappa1; //6.0;
//  R1 = R2 = 8.0;// 6*max(ap,am) + 6.0/kappa1;
//  H = 8.0*kappa1;//3.0*kappa1; //6.0;

  Nr = 50;
  Nz = 50;

}

void CoulombFluid::SetName(char *name) {

  strcpy(project, name);

}

void CoulombFluid::TestFunction() {

  int l,N = 300;
  double *n;
  time_t start,end;

  n = (double*) calloc (N*N*N,sizeof(double));
  h.assign(1.0,N,N,N);

  time (&start); cout << "\n** Running project **\n" << endl;

  for (l=0;l<100;l++){
  for (i=0;i<N;i++){    for (j=0;j<N;j++){      for (k=0;k<N;k++){
    h[i][j][k] = i*j*k;
  }}}
  }

  time (&end);
  cout << "\nProject ran for " << setprecision (0) << int(difftime (end,start)/3600.0) << " hours, " << int(int(difftime (end,start))%3600/60.0) << " minutes, and " << int(difftime (end,start))%60 << " seconds" << endl;

  time (&start); cout << "\n** Running project **\n" << endl;

  for (l=0;l<5;l++){
  for (i=0;i<N;i++){    for (j=0;j<N;j++){      for (k=0;k<N;k++){
    h[i][j][k] = i*j*k;
  }}}
  }

  time (&end);
  cout << "\nProject ran for " << setprecision (0) << int(difftime (end,start)/3600.0) << " hours, " << int(int(difftime (end,start))%3600/60.0) << " minutes, and " << int(difftime (end,start))%60 << " seconds" << endl;

  time (&start); cout << "\n** Running project **\n" << endl;


  for (l=0;l<100;l++){
  for (i=0;i<N;i++){    for (j=0;j<N;j++){      for (k=0;k<N;k++){
    n[i+j*N+k*N*N] = i*j*k;
  }}}
  }
  time (&end);
  cout << "\nProject ran for " << setprecision (0) << int(difftime (end,start)/3600.0) << " hours, " << int(int(difftime (end,start))%3600/60.0) << " minutes, and " << int(difftime (end,start))%60 << " seconds" << endl;
  cout << "Conclusion: use the Mat3D type, instead of a double vector. It's faster, and does not use one long array of memory" << endl;


}

void CoulombFluid::TestFunction2() {

  int N = 5;
  time_t start,end;

  h.assign(1.0,N,N,N);
  c.assign(1.0,N,N,N);

  time (&start); cout << "\n** Running project **\n" << endl;

  for (i=0;i<N;i++){    for (j=0;j<N;j++){      for (k=0;k<N;k++){
    h[i][j][k] = i*j*k; c[i][j][k] = -i*j*k;
  }}}

  for (i=0;i<N;i++){    for (j=0;j<N;j++){      for (k=0;k<N;k++){
    cout << setw(3) << h[i][j][k]-c[i][j][k];
  } cout << setw(5) << ""; } cout << endl;}

  time (&end);
  cout << "\nProject ran for " << setprecision (0) << int(difftime (end,start)/3600.0) << " hours, " << int(int(difftime (end,start))%3600/60.0) << " minutes, and " << int(difftime (end,start))%60 << " seconds" << endl;
```

```
}

/*void CoulombFluid::TestFunction3() {

  // Hankel transformation test
 double *htrans, *ctrans;
  int l,N = 100;
  time_t start,end;

  htrans = (double*) calloc (N, sizeof(double));
  ctrans = (double*) calloc (N, sizeof(double));

  h.assign(1.0,N,N,N);  c.assign(1.0,N,N,N);
  hFour.assign(1.0,N,N,N);  cFour.assign(1.0,N,N,N);

  time (&start); cout << "\n** Running project **\n" << endl;

  for (j=0;i<N;i++){ for (k=0;i<N;i++){

  for (i=0;i<N;i++){
    htrans[i] = h[i][j][k]; ctrans[i] = c[i][j][k];
  }

  BesselTransform(htrans);  BesselTransform(ctrans);

  for (i=0;i<N;i++){
    hFour[i][j][k] = htrans[i]; cFour[i][j][k] = ctrans[i];
  }

  }}

  dataname = (char*) calloc (strlen(projectname)+14,sizeof(char));
  strcpy (dataname,"Data/"); strcat(dataname,projectname); strcat(dataname,"_phi.dat");

  cout << "\n\nfile is named " << dataname << endl;

  datafile.open(dataname);

  datafile << "# Mean potential phi for a system with characteristics:" << endl;
  datafile << "#" << setw(40) << "Number of grid points N = " << N << setw(40) << "concentration cs1 = " << cs1 << " M " << endl;
  datafile << "#" << setw(40) << "radius of cations ap = " << ap << " nm" << setw(40) << "Bjerrum length of lB1 = " << lB1 << " nm" << endl;
  datafile << "#" << setw(40) << "Debye length 1/kappaD1 = " << 1.0/kappa1 << " nm" << setw(40) << endl;

  datafile << "\n#" << setw(20) << "r" << setw(20) << "h(r)" << setw(20) << "h(k)" << endl;

  for (i=1;i<N;i++) datafile << setw(20) << r[i] << setw(20) << h[i][0][0] << setw(20) << hFour[i][0][0] << endl;

  datafile.close();

  time (&end);
  cout << "\nProject ran for " << setprecision (0) << int(difftime (end,start)/3600.0) << " hours, " << int(int(difftime (end,start))%3600/60.0) << " minutes, and " << int(difftime (end,start))%60 << " seconds" << endl;
}
*/

void CoulombFluid::SetGrid() {

  i=-1;
  while (i<Nz/2) {
    i++;
    z[i] = H/kappa1 * (-1.0 + i*2.0/Nz + 1.0/Nz);
  }
  while (i<Nz) {
    i++;
//    z[i] = H/kappa2 * (-1.0 + i*2.0/Nz) + H/kappa1/Nz;
    z[i] = H/kappa1 * (-1.0 + i*2.0/Nz) + H/kappa1/Nz;
  }

  for (i=0;i<Nr;i++) r[i] = i*1.0/Nr *R2;
  r[0] = 0.01/Nr *R2;

  for (i=0;i<Nz-1;i++) dz[i] = z[i+1]-z[i];

  dz[Nz-1] = dz[Nz-2];

  i=0; while (z[i]<0) i++;  interface = i;
  epsilon = (z[interface] + z[interface-1]);
  cout << "Interface is defined at z = " << 0.5*epsilon << " nm, after grid point " << interface+1 << " at z = " << z[interface] << " nm." << endl;

  gsl_dht * t = gsl_dht_new(Nr, 0.0, r[Nr-1]);

  for (i=0;i<Nr;i++) kFT[i] = gsl_dht_k_sample(t, i);


//  for (j=0;j<Nz;j++) { if(z[j]<0) {qpv[j]=2.0; qmv[j]=2.0; apv[j]=0.2125; amv[j]=0.2125;} else {qpv[j]=1.0; qmv[j]=1.0; apv[j]=0.425; amv[j]=0.425;} }

}

void CoulombFluid::SetGrid2Walls() {

  i=-1;
  while (i<Nz) {
    i++;
    z[i] = H/kappa1/2.0 * (-1.0 + i*2.0/(Nz-1));
  }

  for (i=0;i<Nr;i++) r[i] = i*1.0/Nr *R1;
  r[0] = 0.01/Nr *R1;

  for (i=0;i<Nz-1;i++) dz[i] = z[i+1]-z[i];

  dz[Nz-1] = dz[Nz-2]; //dz[0] = dz[Nz-1] = dz[1] = dz[Nz-2] = ap;
//  z[1] = z[2] - ap;  z[0] = z[1] - ap;
```

```
//  z[Nz-2] = z[Nz-3] + ap;  z[Nz-1] = z[Nz-2] + ap;

  gsl_dht * t = gsl_dht_new(Nr, 0.0, r[Nr-1]);

  for (i=0;i<Nr;i++) kFT[i] = gsl_dht_k_sample(t, i);

  for (j=0;j<Nz;j++)    { n[j] = cp[j]*dz[j]; }    for (j=Nz;j<2*Nz;j++) { n[j] = cm[j-Nz]*dz[j-Nz]; }

}

void CoulombFluid::SetQDNA(double LDNA) {

  for(i=0;i<Nz;i++) QDNA[i] = 0;
  i=0;
  while (i<Nz and fabs(z[i]-z[0]) < LDNA) {
    QDNA[i] = sigmaDNA*6.0;
    i++;
  }
  i=Nz-1;
  while (i>0 and fabs(z[Nz-1]-z[i]) < LDNA) {
    QDNA[i] += sigmaDNA*6.0;
    i--;
  }
  //for(i=0;i<Nz;i++) cout << z[i] << setw(15) << QDNA[i] << endl;

}

void CoulombFluid::SetGrid2Walls1Dielectric() {

  i=-1;
  while (i<Nz-1) {
    i++;
    z[i] = (i+1)*H/kappa1/(Nz-1)+ 0.2;
  }

  for (i=0;i<Nr;i++) r[i] = i*1.0/Nr *R1;
  r[0] = 0.01/Nr *R1;
  if (2*ap < r[2]) r[1] = 2.001*ap;

  for (i=0;i<Nz-1;i++) dz[i] = z[i+1]-z[i];

  dz[Nz-1] = dz[Nz-2]; //dz[0] = dz[Nz-1] = dz[1] = dz[Nz-2] = ap;
//  z[1] = z[2] - ap;  z[0] = z[1] - ap;
//  z[Nz-2] = z[Nz-3] + ap;  z[Nz-1] = z[Nz-2] + ap;

  gsl_dht * t = gsl_dht_new(Nr, 0.0, r[Nr-1]);

  for (i=0;i<Nr;i++) kFT[i] = gsl_dht_k_sample(t, i);

  epsilon = 0;

  for (i=0;i<Nr;i++){ Theta[i] = 2*pi*period*gsl_sf_bessel_J1(period*kFT[i])/kFT[i]; }
  for (i=0;i<Nr;i++){ ThetaGauss[i] = max(exp(-kFT[i]*kFT[i]/(4*pi)/period) / period , 1e-200); }
}

void CoulombFluid::SetGrid2WallSession(double NzMem) {

  double zmean, step1, step2;
  int ib;

  step1 = H0/kappa1 - 2.0*Lw*(1.0-exp(depth/2.0));
  step1 /= 1.0*(NzMem-1)  * 0.9;
  step2 = step1*exp(-depth/2.0);

  Nl = Lw/step2; cout << "Nl = " << Nl << " step1 = " << step1 << " and step2 = " << step2 << " H0/kappa1 = " << H0/kappa1 << endl;

  i=0; z[0] = -H/kappa1/2.0;
  while (z[i]<(z[0]+Lw) and i<NzMem-1) {
    i++;
    z[i] = z[0] + i*step2;  //H0/kappa1*exp(-depth/2.0)/2.0 * (-1.0 + i*2.0/(NzMem-1));
  }
  ib=i; Nl = i;
  while (z[i]<z[0]+H/kappa1-Lw) {
    i++;
    z[i] = z[ib] + (i-ib)*step1;
  }
  ib=i;
  while (i-ib < Nl) {
    i++;
    z[i] = z[ib] + (i-ib)*step2;
  }
  Nz=i+1; cout << "Nl = " << Nl << " and Nz = " << Nz << endl;
  if (Nz>NzMem) {
  zmean = (z[Nz-1]+z[0])/2.0;
  for (i=0;i<Nz;i++) z[i] -= zmean;


  for (i=0;i<Nr;i++) r[i] = i*1.0/Nr *R1;
  r[0] = 0.01/Nr *R1;

//  for (i=1;i<Nz-1;i++) dz[i] = (z[i+1]-z[i-1])/2.0;
  for (i=1;i<Nz/2;i++) dz[i] = z[i+1]-z[i];  for (i=Nz/2;i<Nz-1;i++) dz[i] = z[i]-z[i-1];

  dz[Nz-1] = dz[Nz-2];  dz[0] = dz[1]; //dz[0] = dz[Nz-1] = dz[1] = dz[Nz-2] = ap;
//  z[1] = z[2] - ap;  z[0] = z[1] - ap;
//  z[Nz-2] = z[Nz-3] + ap;  z[Nz-1] = z[Nz-2] + ap;

  gsl_dht * t = gsl_dht_new(Nr, 0.0, r[Nr-1]);

  for (i=0;i<Nr;i++) kFT[i] = gsl_dht_k_sample(t, i);

}

void CoulombFluid::SetGridExcluded() {
```

```
  i=-1;
  while (i<Nz/2) {
    i++;
    z[i] = H/kappa1 * (-1.0 + i*2.0/Nz) - aReg/2;
  }
  while (i<Nz) {
    i++;
    z[i] = H/kappa1 * (-1.0 + i*2.0/Nz) + aReg/2;
  }

for (j=0;j<Nz;j++) { if(j<Nz/2) {qpv[j]=2.0; qmv[j]=2.0; apv[j]=0.2125; amv[j]=0.2125;} else {qpv[j]=1.0; qmv[j]=1.0; apv[j]=0.425; amv[j]=0.425;} }

/* Alternative grid */
  i=0;
  while (i<Nz/2) {
    z[i] = 4*apv[i] * (-Nz/2 + i + 0.5);
    i++;
  }
  while (i<Nz) {
    z[i] = 4*apv[i-Nz/2] * (-Nz/2 + i + 0.5);
    i++;
  }


  for (i=0;i<Nr;i++) r[i] = i*1.0/Nr *R1;
  r[0] = 0.01/Nr *R1;

  for (i=1;i<Nz-1;i++) dz[i] = 0.5*(z[i+1]-z[i-1]);

  dz[0] = dz[1];
  dz[Nz-1] = dz[Nz-2];

  i=0; while (z[i]<0) i++;  interface = i;
  epsilon = (z[interface] + z[interface-1]);

}

void CoulombFluid::InformationFile(bool ask) {

  ifstream check;
  ofstream datafile;
  char filename[70],answer;

  strcpy(filename,"Data/"); strcat(filename, project); strcat(filename,"_Info.dat");

  if (ask) {
    check.open (filename, ios::binary );
    check.seekg (0, ios::end);
    if (check.tellg()>1) {
      cout << "File \"" << filename << "\" already contains information. Overwrite (o) or exit (e)?" << endl;
      answer = cin.get ();
      if (answer != 'o') exit(1);
    }
    check.close();
  }

  datafile.open(filename,ios::trunc);
  datafile << "The " << project << "*.dat files were obtained by using the following parameters:\n" << endl;
  datafile << setw(30) << "Bjerrum length phase 1" << setw(10) << "lB1" << setw(10) << lB1 << setw(4) << "nm" << setw(40) << "Bjerrum length phase 2" << setw(10) << "lB2" << setw(14) << lB2 << setw(4) << "nm" << endl;
  datafile << setw(30) << "concentration cations phase 1" << setw(10) << "cs1" << setw(10) << cs1/NA << setw(4) << "M" << setw(40) << "concentration cations phase 2" << setw(10) << "cs2" << setw(14) << cs2/NA << setw(4) << "M" << endl;
  datafile << setw(30) << "radius cations phase 1" << setw(10) << "ap1" << setw(10) << apv[0] << setw(4) << "nm" << setw(40) << "radius anions phase 1" << setw(10) << "am1" << setw(14) << amv[0] << setw(4) << "nm" << endl;
  datafile << setw(30) << "charge cations phase 1" << setw(10) << "qp1" << setw(10) << qpv[0] << setw(4) << "e" << setw(40) << "charge anions phase 1" << setw(10) << "qm1" << setw(14) << qmv[0] << setw(4) << "e" << endl;
  datafile << setw(30) << "radius cations phase 2" << setw(10) << "ap2" << setw(10) << apv[Nz-1] << setw(4) << "nm" << setw(40) << "radius anions phase 2" << setw(10) << "am2" << setw(14) << amv[Nz-1] << setw(4) << "nm" << endl;
  datafile << setw(30) << "charge cations phase 2" << setw(10) << "qp2" << setw(10) << qpv[Nz-1] << setw(4) << "e" << setw(40) << "charge anions phase 2" << setw(10) << "qm2" << setw(14) << qmv[Nz-1] << setw(4) << "e" << endl;
  datafile << setw(30) << "Debye length phase 1" << setw(10) << "1/kappa1" << setw(10) << 1.0/kappa1 << setw(4) << "nm" << setw(40) << "Debye length phase 2" << setw(10) << "1/kappa2" << setw(14) << 1.0/kappa2 << setw(4) << "nm" << endl;
  datafile << setw(30) << "relative permittivity phase 1" << setw(10) << "e1" << setw(10) << e1 << setw(4) << " " << setw(40) << "relative permittivity phase 2" << setw(10) << "e2" << setw(14) << e2 << setw(4) << " \n" << endl;
  datafile << setw(30) << "width of one phase" << setw(10) << "H" << setw(10) << H << " Debye lengths" << setw(50) << "Mixing parameter alpha = " << alpha << endl;
  datafile << setw(30) << "maximal radial component" << setw(10) << "R1" << setw(10) << R1 << " nm" << endl;

  cout << "\n\nThe program with name \"" << project << "\" is running using the following parameters:\n" << endl;
  cout << setw(30) << "Bjerrum length phase 1" << setw(10) << "lB1" << setw(10) << lB1 << setw(4) << "nm" << setw(40) << "Bjerrum length phase 2" << setw(10) << "lB2" << setw(14) << lB2 << setw(4) << "nm" << endl;
  cout << setw(30) << "concentration cations phase 1" << setw(10) << "cs1" << setw(10) << cs1/NA << setw(4) << "M" << setw(40) << "concentration cations phase 2" << setw(10) << "cs2" << setw(14) << cs2/NA << setw(4) << "M" << endl;
  cout << setw(30) << "radius cations phase 1" << setw(10) << "ap1" << setw(10) << apv[0] << setw(4) << "nm" << setw(40) << "radius anions phase 1" << setw(10) << "am1" << setw(14) << amv[0] << setw(4) << "nm" << endl;
  cout << setw(30) << "charge cations phase 1" << setw(10) << "qp1" << setw(10) << qpv[0] << setw(4) << "e" << setw(40) << "charge anions phase 1" << setw(10) << "qm1" << setw(14) << qmv[0] << setw(4) << "e" << endl;
  cout << setw(30) << "radius cations phase 2" << setw(10) << "ap2" << setw(10) << apv[Nz-1] << setw(4) << "nm" << setw(40) << "radius anions phase 2" << setw(10) << "am2" << setw(14) << amv[Nz-1] << setw(4) << "nm" << endl;
  cout << setw(30) << "charge cations phase 2" << setw(10) << "qp2" << setw(10) << qpv[Nz-1] << setw(4) << "e" << setw(40) << "charge anions phase 2" << setw(10) << "qm2" << setw(14) << qmv[Nz-1] << setw(4) << "e" << endl;
  cout << setw(30) << "Debye length phase 1" << setw(10) << "1/kappa1" << setw(10) << 1.0/kappa1 << setw(4) << "nm" << setw(40) << "Debye length phase 2" << setw(10) << "1/kappa2" << setw(14) << 1.0/kappa2 << setw(4) << "nm" << endl;
  cout << setw(30) << "relative permittivity phase 1" << setw(10) << "e1" << setw(10) << e1 << setw(4) << " " << setw(40) << "relative permittivity phase 2" << setw(10) << "e2" << setw(14) << e2 << setw(4) << " \n" << endl;
  cout << setw(30) << "width of one phase" << setw(10) << "H" << setw(10) << H << " Debye lengths" << setw(50) << "Mixing parameter alpha = " << alpha << endl;
  cout << setw(30) << "maximal radial component" << setw(10) << "R1" << setw(10) << R1 << " nm" << endl;

  datafile.close();

}

void CoulombFluid::SetPairPotential() {

  double image, screen;
  double a0,b0,H1,H2,DV;

  screen = 2*lB1*lB2/(lB2+lB1);
  image = (lB2-lB1)/(lB2+lB1);

  H1 = epsilon-z[0]; H2 = z[Nz-1]-epsilon; DV = 0.0; a0=DV/(H1+lB2/lB1*H2); b0=DV/(lB1/lB2*H1+H2);
  /* External field */
  for (j=0;j<Nz;j++) {
    if (z[j] < 0.0) {
      Vextp[j] =  qpv[j]*qpv[j]*lB1/fabs(2*z[j]-epsilon)*image;
      Vextm[j] =  qmv[j]*qmv[j]*lB1/fabs(2*z[j]-epsilon)*image;
    }
    if (z[j] >= 0.0) {
      Vextp[j] = -qpv[j]*qpv[j]*lB2/fabs(2*z[j]-epsilon)*image;
      Vextm[j] = -qmv[j]*qmv[j]*lB2/fabs(2*z[j]-epsilon)*image;
    }
```

```
  //    if (fabs(z[j]-0.5*epsilon) < aReg) { Vextp[j] = 40.0; Vextm[j] = 40.0; }
    if (fabs(z[j]-0.5*epsilon) < apv[j]) {
    Vextp[j] = 0.5*lB2*qpv[j]*qpv[j]/apv[j]/(2*apv[j] - epsilon)*image*(z[j]+apv[j])  -  0.5*lB1*qpv[j]*qpv[j]/apv[j]/(2*apv[j] - epsilon)*image*(apv[j]-z[j]);
    }
    if (fabs(z[j]-0.5*epsilon) < amv[j]) {
    Vextm[j] = 0.5*lB2*qmv[j]*qmv[j]/amv[j]/(2*amv[j] - epsilon)*image*(z[j]+amv[j])  -  0.5*lB1*qmv[j]*qmv[j]/amv[j]/(2*amv[j] - epsilon)*image*(amv[j]-z[j]);
    }
  }
    /* Solvation energy */
/*    if (fabs(z[j]) < apv[j]) {
      Vextp[j] += pow(fabs(z[j])-apv[j],2.0)*4.0/pow(apv[j],2.0); //0.0*(lB2-lB1)*qpv[j]*qpv[j]/apv[j];
      Vextm[j] += pow(fabs(z[j])-amv[j],2.0)*4.0/pow(amv[j],2.0); //-0.0*(lB2-lB1)*qmv[j]*qmv[j]/amv[j];
    }
    if (z[j] >= 0.0) {      Vextp[j] += -qpv[j]*b0*(z[j]-z[0]);      Vextm[j] += +qmv[j]*b0*(z[j]-z[0]);    }
    else {      Vextp[j] += -qpv[j]*a0*(z[j]-z[0]);      Vextm[j] += +qmv[j]*a0*(z[j]-z[0]);    }*/

  for (j=0;j<Nz;j++) {
    phi[j] = - Vextp[j] / qpv[j];
  }

  /* Positive particles are counted from j,k = 0 to N - 1; negative particles from j,k = N to 2N - 1. */
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (j==k and r[i]==0) { u[i][j][k] = Large; continue; }
    if (z[j]<0.0 and z[k]<0.0)  {u[i][j][k] = qpv[j]*qpv[j]*lB1/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)) + qpv[j]*qpv[j]*lB1/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image; continue;}
    if (z[j]>=0.0 and z[k]>=0.0){u[i][j][k] = qpv[j]*qpv[j]*lB2/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)) + qpv[j]*qpv[j]*lB2/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image; continue;}
    if (z[j]<0.0 and z[k]>=0.0) {u[i][j][k] = qpv[j]*qpv[k]*screen/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)); continue;}
    if (z[j]>=0.0 and z[k]<0.0) {u[i][j][k] = qpv[j]*qpv[k]*screen/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));}
  }}}
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (j==k and r[i]==0) { u[i][j][k] = -Large; continue; }
    if (z[j]<0.0 and z[k]<0.0)  {u[i][j+Nz][k] = -qpv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)) - qpv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image; continue;}
    if (z[j]>=0.0 and z[k]>=0.0){u[i][j+Nz][k] = -qpv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)) - qpv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image; continue;}
    if (z[j]<0.0 and z[k]>=0.0) {u[i][j+Nz][k] = -qpv[j]*qmv[k]*screen/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)); continue;}
    if (z[j]>=0.0 and z[k]<0.0) {u[i][j+Nz][k] = -qpv[j]*qmv[k]*screen/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));}
  }}}
  for (i=0;i<Nr;i++){    for (j=0.0;j<Nz;j++){      for (k=0;k<Nz;k++){
    u[i][j][k+Nz] = u[i][j+Nz][k];
  }}}
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (j==k and r[i]==0) { u[i][j+Nz][k+Nz] = Large; continue; }
    if (z[j]<0.0 and z[k]<0.0)  {u[i][j+Nz][k+Nz] = qmv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)) + qmv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image; continue;}
    if (z[j]>=0.0 and z[k]>=0.0){u[i][j+Nz][k+Nz] = qmv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)) + qmv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image; continue;}
    if (z[j]<0.0 and z[k]>=0.0) {u[i][j+Nz][k+Nz] = qmv[j]*qmv[k]*screen/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)); continue;}
    if (z[j]>=0.0 and z[k]<0.0) {u[i][j+Nz][k+Nz] = qmv[j]*qmv[k]*screen/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));}
  }}}

  /* Positive particles are counted from j,k = 0 to N - 1; negative particles from j,k = N to 2N - 1. */

  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    uReg[i][j][k] = u[i][j][k];
    if (fabs(z[j]-z[k]) < aReg and (fabs(z[j]-0.5*epsilon) + fabs(z[k]-0.5*epsilon)) >= aReg) {
      if (z[j]<0.0 and z[k]<0.0)    {uReg[i][j][k] = qpv[j]*qpv[j]*lB1/sqrt(r[i]*r[i] + pow(aReg,2.0)) + qpv[j]*qpv[j]*lB1/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image;}
      if (z[j]>=0.0 and z[k]>=0.0)  {uReg[i][j][k] = qpv[j]*qpv[j]*lB2/sqrt(r[i]*r[i] + pow(aReg,2.0)) + qpv[j]*qpv[j]*lB2/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image;}
    }
    if (fabs(z[j]-z[k]) < aReg and (fabs(z[j]-0.5*epsilon) + fabs(z[k]-0.5*epsilon)) < aReg) {
      if (z[j]<0.0 and z[k]<0.0)    {uReg[i][j][k] = qpv[j]*qpv[j]*lB1/sqrt(r[i]*r[i] + pow(aReg,2.0)) + qpv[j]*qpv[j]*lB1/sqrt(r[i]*r[i] + pow(aReg,2.0))*image;}
      if (z[j]>=0.0 and z[k]>=0.0)  {uReg[i][j][k] = qpv[j]*qpv[j]*lB2/sqrt(r[i]*r[i] + pow(aReg,2.0)) + qpv[j]*qpv[j]*lB2/sqrt(r[i]*r[i] + pow(aReg,2.0))*image;}
      if (z[j]<0.0 and z[k]>=0.0)   {uReg[i][j][k] = qpv[j]*qpv[k]*screen/sqrt(r[i]*r[i] + pow(aReg,2.0));}
      if (z[j]>=0.0 and z[k]<0.0)   {uReg[i][j][k] = qpv[j]*qpv[k]*screen/sqrt(r[i]*r[i] + pow(aReg,2.0));}
    }
  }}}

  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    uReg[i][j+Nz][k] = u[i][j+Nz][k];
    if (fabs(z[j]-z[k]) < aReg and (fabs(z[j]-0.5*epsilon) + fabs(z[k]-0.5*epsilon)) >= aReg) {
      if (z[j]<0.0 and z[k]<0.0)    {uReg[i][j+Nz][k] = -qpv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(aReg,2.0)) - qpv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image;}
      if (z[j]>=0.0 and z[k]>=0.0)  {uReg[i][j+Nz][k] = -qpv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(aReg,2.0)) - qpv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image;}
    }
    if (fabs(z[j]-z[k]) < aReg and (fabs(z[j]-0.5*epsilon) + fabs(z[k]-0.5*epsilon)) < aReg) {
      if (z[j]<0.0 and z[k]<0.0)    {uReg[i][j+Nz][k] = -qpv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(aReg,2.0)) - qpv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(aReg,2.0))*image;}
      if (z[j]>=0.0 and z[k]>=0.0)  {uReg[i][j+Nz][k] = -qpv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(aReg,2.0)) - qpv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(aReg,2.0))*image;}
      if (z[j]<0.0 and z[k]>=0.0)   {uReg[i][j+Nz][k] = -qpv[j]*qmv[k]*screen/sqrt(r[i]*r[i] + pow(aReg,2.0));}
      if (z[j]>=0.0 and z[k]<0.0)   {uReg[i][j+Nz][k] = -qpv[j]*qmv[k]*screen/sqrt(r[i]*r[i] + pow(aReg,2.0));}
    }
    uReg[i][j][k+Nz] = uReg[i][j+Nz][k];
  }}}

  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    uReg[i][j+Nz][k+Nz] = u[i][j+Nz][k+Nz];
    if (fabs(z[j]-z[k]) < aReg and (fabs(z[j]-0.5*epsilon) + fabs(z[k]-0.5*epsilon)) >= aReg) {
      if (z[j]<0.0 and z[k]<0.0)    {uReg[i][j+Nz][k+Nz] = qmv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(aReg,2.0)) + qmv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image;}
      if (z[j]>=0.0 and z[k]>=0.0)  {uReg[i][j+Nz][k+Nz] = qmv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(aReg,2.0)) + qmv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image;}
    }
    if (fabs(z[j]-z[k]) < aReg and (fabs(z[j]-0.5*epsilon) + fabs(z[k]-0.5*epsilon)) < aReg) {
      if (z[j]<0.0 and z[k]<0.0)    {uReg[i][j+Nz][k+Nz] = qmv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(aReg,2.0)) + qmv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(aReg,2.0))*image;}
      if (z[j]>=0.0 and z[k]>=0.0)  {uReg[i][j+Nz][k+Nz] = qmv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(aReg,2.0)) + qmv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(aReg,2.0))*image;}
      if (z[j]<0.0 and z[k]>=0.0)   {uReg[i][j+Nz][k+Nz] = qmv[j]*qmv[k]*screen/sqrt(r[i]*r[i] + pow(aReg,2.0));}
      if (z[j]>=0.0 and z[k]<0.0)   {uReg[i][j+Nz][k+Nz] = qmv[j]*qmv[k]*screen/sqrt(r[i]*r[i] + pow(aReg,2.0));}
    }
  }}}

}

void CoulombFluid::SetPairPotential2Walls1Dielectric() {

  double image, screen;
  double a0,b0,H1,H2,DV;

  screen = 2*lB1*lB2/(lB2+lB1);
  image = (lB2-lB1)/(lB2+lB1);

  H1 = epsilon-z[0]; H2 = z[Nz-1]-epsilon; DV = 0.0; a0=DV/(H1+lB2/lB1*H2); b0=DV/(lB1/lB2*H1+H2);
  /* External field */
  for (j=0;j<Nz;j++) {
    if (z[j] < 0.0) {
```

```
      Vextp[j] =  qpv[j]*qpv[j]*lB1/fabs(2*z[j]-epsilon)*image;
      Vextm[j] =  qmv[j]*qmv[j]*lB1/fabs(2*z[j]-epsilon)*image;
    }
    if (z[j] >= 0.0) {
      Vextp[j] = -qpv[j]*qpv[j]*lB2/fabs(2*z[j]-epsilon)*image;
      Vextm[j] = -qmv[j]*qmv[j]*lB2/fabs(2*z[j]-epsilon)*image
    }
    if (z[j] < apv[j]) Vextp[j] += 3.0*pow((apv[j]-z[j])/apv[j],2.0);
    if (z[j] < amv[j]) Vextm[j] += 3.0*pow((amv[j]-z[j])/amv[j],2.0);
//    if (z[j] > z[Nz-1]-1.0) Vextp[j] += 6.0*(1.0 +z[j] - z[Nz-1]);
//    if (z[j] > z[Nz-1]-1.0) Vextm[j] += 6.0*(1.0 +z[j] - z[Nz-1]);
  //    if (fabs(z[j]-0.5*epsilon) < aReg) { Vextp[j] = 40.0; Vextm[j] = 40.0; }
/*    if (fabs(z[j]-0.5*epsilon) < apv[j]) {
      if (z[j]>= 0.0) {
    Vextp[j] = 0.5*lB2*qpv[j]*qpv[j]/apv[j]/(2*apv[j] - epsilon)*image*z[j];
    Vextm[j] = 0.5*lB2*qmv[j]*qmv[j]/amv[j]/(2*amv[j] - epsilon)*image*z[j];
      }
      else {
    Vextp[j] = 0.5*lB1*qpv[j]*qpv[j]/apv[j]/(-2*apv[j] - epsilon)*image*z[j];
    Vextm[j] = 0.5*lB1*qmv[j]*qmv[j]/amv[j]/(-2*amv[j] - epsilon)*image*z[j];
      }
    }*/
    /* Solvation energy */
/*    if (fabs(z[j]) < apv[j]) {
      Vextp[j] += pow(fabs(z[j])-apv[j],2.0)/pow(apv[j],2.0)*fabs(lB2-lB1)*qpv[j]*qpv[j]/apv[j];
      Vextm[j] += pow(fabs(z[j])-amv[j],2.0)/pow(amv[j],2.0)*fabs(lB2-lB1)*qmv[j]*qmv[j]/amv[j];
    }*/
//    if (z[j] >= 0.0) {      Vextp[j] += -qpv[j]*b0*(z[j]-z[0]);      Vextm[j] += +qmv[j]*b0*(z[j]-z[0]);    }
//    else {      Vextp[j] += -qpv[j]*a0*(z[j]-z[0]);      Vextm[j] += +qmv[j]*a0*(z[j]-z[0]);    }
  }
//  for (j=0;j<Nz;j++) {
//    phi[j] = - Vextp[j] / qpv[j];
//  }

  /* Positive particles are counted from j,k = 0 to N - 1; negative particles from j,k = N to 2N - 1. */
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (j==k and r[i]==0) { u[i][j][k] = Large; continue; }
    if (z[j]<0.0 and z[k]<0.0)  {u[i][j][k] = qpv[j]*qpv[j]*lB1/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)) + qpv[j]*qpv[j]*lB1/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image; continue;}
    if (z[j]>=0.0 and z[k]>=0.0){u[i][j][k] = qpv[j]*qpv[j]*lB2/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)) + qpv[j]*qpv[j]*lB2/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image; continue;}
    if (z[j]<0.0 and z[k]>=0.0) {u[i][j][k] = qpv[j]*qpv[k]*screen/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)); continue;}
    if (z[j]>=0.0 and z[k]<0.0) {u[i][j][k] = qpv[j]*qpv[k]*screen/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));}
  }}}
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (j==k and r[i]==0) { u[i][j][k] = -Large; continue; }
    if (z[j]<0.0 and z[k]<0.0)  {u[i][j+Nz][k] = -qpv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)) - qpv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image; continue;}
    if (z[j]>=0.0 and z[k]>=0.0){u[i][j+Nz][k] = -qpv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)) - qpv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image; continue;}
    if (z[j]<0.0 and z[k]>=0.0) {u[i][j+Nz][k] = -qpv[j]*qmv[k]*screen/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)); continue;}
    if (z[j]>=0.0 and z[k]<0.0) {u[i][j+Nz][k] = -qpv[j]*qmv[k]*screen/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));}
  }}}
  for (i=0;i<Nr;i++){    for (j=0.0;j<Nz;j++){      for (k=0;k<Nz;k++){
    u[i][j][k+Nz] = u[i][j+Nz][k];
  }}}
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (j==k and r[i]==0) { u[i][j+Nz][k+Nz] = Large; continue; }
    if (z[j]<0.0 and z[k]<0.0)  {u[i][j+Nz][k+Nz] = qmv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)) + qmv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image; continue;}
    if (z[j]>=0.0 and z[k]>=0.0){u[i][j+Nz][k+Nz] = qmv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)) + qmv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image; continue;}
    if (z[j]<0.0 and z[k]>=0.0) {u[i][j+Nz][k+Nz] = qmv[j]*qmv[k]*screen/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)); continue;}
    if (z[j]>=0.0 and z[k]<0.0) {u[i][j+Nz][k+Nz] = qmv[j]*qmv[k]*screen/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));}
  }}}

  /* Positive particles are counted from j,k = 0 to N - 1; negative particles from j,k = N to 2N - 1. */

  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    uReg[i][j][k] = u[i][j][k];
    if (fabs(z[j]-z[k]) < aReg and (fabs(z[j]-0.5*epsilon) + fabs(z[k]-0.5*epsilon)) >= aReg) {
      if (z[j]<0.0 and z[k]<0.0)    {uReg[i][j][k] = qpv[j]*qpv[j]*lB1/sqrt(r[i]*r[i] + pow(aReg,2.0)) + qpv[j]*qpv[j]*lB1/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image;}
      if (z[j]>=0.0 and z[k]>=0.0)  {uReg[i][j][k] = qpv[j]*qpv[j]*lB2/sqrt(r[i]*r[i] + pow(aReg,2.0)) + qpv[j]*qpv[j]*lB2/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image;}
    }
    if (fabs(z[j]-z[k]) < aReg and (fabs(z[j]-0.5*epsilon) + fabs(z[k]-0.5*epsilon)) < aReg) {
      if (z[j]<0.0 and z[k]<0.0)    {uReg[i][j][k] = qpv[j]*qpv[j]*lB1/sqrt(r[i]*r[i] + pow(aReg,2.0)) + qpv[j]*qpv[j]*lB1/sqrt(r[i]*r[i] + pow(aReg,2.0))*image;}
      if (z[j]>=0.0 and z[k]>=0.0)  {uReg[i][j][k] = qpv[j]*qpv[j]*lB2/sqrt(r[i]*r[i] + pow(aReg,2.0)) + qpv[j]*qpv[j]*lB2/sqrt(r[i]*r[i] + pow(aReg,2.0))*image;}
      if (z[j]<0.0 and z[k]>=0.0)   {uReg[i][j][k] = qpv[j]*qpv[k]*screen/sqrt(r[i]*r[i] + pow(aReg,2.0));}
      if (z[j]>=0.0 and z[k]<0.0)   {uReg[i][j][k] = qpv[j]*qpv[k]*screen/sqrt(r[i]*r[i] + pow(aReg,2.0));}
    }
  }}}

  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    uReg[i][j+Nz][k] = u[i][j+Nz][k];
    if (fabs(z[j]-z[k]) < aReg and (fabs(z[j]-0.5*epsilon) + fabs(z[k]-0.5*epsilon)) >= aReg) {
      if (z[j]<0.0 and z[k]<0.0)    {uReg[i][j+Nz][k] = -qpv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(aReg,2.0)) - qpv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image;}
      if (z[j]>=0.0 and z[k]>=0.0)  {uReg[i][j+Nz][k] = -qpv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(aReg,2.0)) - qpv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image;}
    }
    if (fabs(z[j]-z[k]) < aReg and (fabs(z[j]-0.5*epsilon) + fabs(z[k]-0.5*epsilon)) < aReg) {
      if (z[j]<0.0 and z[k]<0.0)    {uReg[i][j+Nz][k] = -qpv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(aReg,2.0)) - qpv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(aReg,2.0))*image;}
      if (z[j]>=0.0 and z[k]>=0.0)  {uReg[i][j+Nz][k] = -qpv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(aReg,2.0)) - qpv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(aReg,2.0))*image;}
      if (z[j]<0.0 and z[k]>=0.0)   {uReg[i][j+Nz][k] = -qpv[j]*qmv[k]*screen/sqrt(r[i]*r[i] + pow(aReg,2.0));}
      if (z[j]>=0.0 and z[k]<0.0)   {uReg[i][j+Nz][k] = -qpv[j]*qmv[k]*screen/sqrt(r[i]*r[i] + pow(aReg,2.0));}
    }
    uReg[i][j][k+Nz] = uReg[i][j+Nz][k];
  }}}

  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    uReg[i][j+Nz][k+Nz] = u[i][j+Nz][k+Nz];
    if (fabs(z[j]-z[k]) < aReg and (fabs(z[j]-0.5*epsilon) + fabs(z[k]-0.5*epsilon)) >= aReg) {
      if (z[j]<0.0 and z[k]<0.0)    {uReg[i][j+Nz][k+Nz] = qmv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(aReg,2.0)) + qmv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image;}
      if (z[j]>=0.0 and z[k]>=0.0)  {uReg[i][j+Nz][k+Nz] = qmv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(aReg,2.0)) + qmv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(z[j]+z[k]-epsilon,2.0))*image;}
    }
    if (fabs(z[j]-z[k]) < aReg and (fabs(z[j]-0.5*epsilon) + fabs(z[k]-0.5*epsilon)) < aReg) {
      if (z[j]<0.0 and z[k]<0.0)    {uReg[i][j+Nz][k+Nz] = qmv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(aReg,2.0)) + qmv[j]*qmv[j]*lB1/sqrt(r[i]*r[i] + pow(aReg,2.0))*image;}
      if (z[j]>=0.0 and z[k]>=0.0)  {uReg[i][j+Nz][k+Nz] = qmv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(aReg,2.0)) + qmv[j]*qmv[j]*lB2/sqrt(r[i]*r[i] + pow(aReg,2.0))*image;}
      if (z[j]<0.0 and z[k]>=0.0)   {uReg[i][j+Nz][k+Nz] = qmv[j]*qmv[k]*screen/sqrt(r[i]*r[i] + pow(aReg,2.0));}
      if (z[j]>=0.0 and z[k]<0.0)   {uReg[i][j+Nz][k+Nz] = qmv[j]*qmv[k]*screen/sqrt(r[i]*r[i] + pow(aReg,2.0));}
    }
  }}}
```

```
}

void CoulombFluid::SetPairPotential2Walls() {

  double Vdepthp, Vdepthm;

  Vlength = 5.0/kappa1;//lB1*5.0;
  Vdepthp = -depth;
  Vdepthm = -depth;

  Vlength = 2.1*(z[1]-z[0]);

  sigma = pow(5000,-1.0)*1e2;

  aReg = 5.0/kFT[Nz-1]; cout << "highest value for kFT is " << kFT[Nz-1] << "/nm, and aReg = " << aReg << " nm" << endl;

  /* External field */
  for (j=0;j<Nz;j++) {
    Vextp[j] = Vdepthp*exp(-(z[j]-z[0])/Vlength) + Vdepthp*exp((z[j]-z[Nz-1])/Vlength);
    Vextm[j] = Vdepthm*exp(-(z[j]-z[0])/Vlength) + Vdepthm*exp((z[j]-z[Nz-1])/Vlength);
//    if ( (z[j]-z[0]) < 2*ap or (z[Nz-1]-z[j])< 2*ap) { Vextp[j] = Vdepthp;    Vextm[j] = Vdepthm; } else Vextp[j] = Vextm[j] = 0;
//    Vextp[j] = 0.5*lB1*z[j]*sigma; Vextm[j] = -0.5*lB1*z[j]*sigma;
    if ( j < Nl or (Nz-Nl-1)<j ) { Vextp[j] = Vdepthp;    Vextm[j] = Vdepthm; } else Vextp[j] = Vextm[j] = 0;
//    Vextp[j] = qpv[j]*sigma;
//    Vextm[j] = -Vextp[j];
  }
  for (j=0;j<Nz;j++) {
//    phi[j] = - Vextp[j] / qpv[j];
  }


  /* The Coulomb potential between two point charges */
  /* Positive particles are counted from j,k = 0 to N - 1; negative particles from j,k = N to 2N - 1. */
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (j==k and r[i]==0) {
      u[0][j][k] = Large; continue;
    }
    u[i][j][k] = qp*qp*lB1/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
  }}}
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (j==k and r[i]==0) {
      u[0][j+Nz][k] = -Large; continue;
    }
    u[i][j+Nz][k] = -qp*qm*lB1/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
    u[i][j][k+Nz] = u[i][j+Nz][k];
  }}}
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (j==k and r[i]==0) {
      u[0][j+Nz][k+Nz] = Large; continue;
    }
    u[i][j+Nz][k+Nz] = qm*qm*lB1/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
  }}}

  /* The DHT of the Coulomb potential between two point charges */

  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (r[i] == 0.0) {uHT[i][j][k] = Large; continue;}
    uHT[i][j][k] = qp*qp*lB1*exp(-fabs(z[j]-z[k])*kFT[i])/kFT[i] *2*pi;
  }}}
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=Nz;k<2*Nz;k++){
    if (r[i] == 0.0) {uHT[i][j][k] = -Large; uHT[i][j+Nz][k-Nz] = -Large; continue;}
    uHT[i][j][k] = -qp*qm*lB1*exp(-fabs(z[j]-z[k-Nz])*kFT[i])/kFT[i] *2*pi;
    uHT[i][j+Nz][k-Nz] = uHT[i][j][k];
  }}}
  for (i=0;i<Nr;i++){    for (j=Nz;j<2*Nz;j++){      for (k=Nz;k<2*Nz;k++){
    if (r[i] == 0.0) {uHT[i][j][k] = Large; continue;}
    uHT[i][j][k] = qm*qm*lB1*exp(-fabs(z[j-Nz]-z[k-Nz])*kFT[i])/kFT[i] *2*pi;
  }}}

  /* The regularized Coulomb potential between two point charges */

  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (fabs(z[j]-z[k]) < aReg) {
      uReg[i][j][k] = qp*qp*lB1/sqrt(r[i]*r[i] + aReg*aReg); continue;
    }
    uReg[i][j][k] = u[i][j][k];
  }}}
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (fabs(z[j]-z[k]) < aReg) {
      uReg[i][j][k+Nz] = -qp*qm*lB1/sqrt(r[i]*r[i] + aReg*aReg);
      uReg[i][j+Nz][k] = uReg[i][j][k+Nz];      continue;
    }
    uReg[i][j][k+Nz] = u[i][j][k+Nz];
    uReg[i][j+Nz][k] = uReg[i][j][k+Nz];
  }}}
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (fabs(z[j]-z[k]) < aReg) {
      uReg[i][j+Nz][k+Nz] = qm*qm*lB1/sqrt(r[i]*r[i] + aReg*aReg); continue;
    }
    uReg[i][j+Nz][k+Nz] = u[i][j+Nz][k+Nz];
  }}}

  /* The DHT of the regularized Coulomb potential between two point charges */

  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (r[i] == 0.0) {uRegHT[i][j][k] = Large; continue;}
    if (fabs(z[j]-z[k]) < aReg) {
      uRegHT[i][j][k] = qp*qp*lB1*exp(-aReg*kFT[i])/kFT[i] *2*pi; continue;
    }
    uRegHT[i][j][k] = qp*qp*lB1*exp(-fabs(z[j]-z[k])*kFT[i])/kFT[i] *2*pi;
  }}}
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=Nz;k<2*Nz;k++){
    if (r[i] == 0.0) {uRegHT[i][j][k] = -Large; continue;}
    if (fabs(z[j]-z[k-Nz]) < aReg) {
      uRegHT[i][j][k] = -qp*qm*lB1*exp(-aReg*kFT[i])/kFT[i] *2*pi;
```

```
      uRegHT[i][j+Nz][k-Nz] = uRegHT[i][j][k];      continue;
    }
    uRegHT[i][j][k] = -qp*qm*lB1*exp(-fabs(z[j]-z[k-Nz])*kFT[i])/kFT[i] *2*pi;
    uRegHT[i][j+Nz][k-Nz] = uRegHT[i][j][k];
  }}}
  for (i=0;i<Nr;i++){    for (j=Nz;j<2*Nz;j++){      for (k=Nz;k<2*Nz;k++){
    if (r[i] == 0.0) {uRegHT[i][j][k] = Large; continue;}
    if (fabs(z[j-Nz]-z[k-Nz]) < aReg) {
      uRegHT[i][j][k] = qm*qm*lB1*exp(-aReg*kFT[i])/kFT[i] *2*pi; continue;
    }
    uRegHT[i][j][k] = qm*qm*lB1*exp(-fabs(z[j-Nz]-z[k-Nz])*kFT[i])/kFT[i] *2*pi;
  }}}

  /* The alternative regularized Coulomb potential between two point charges, choice of author (JZ) */
/*
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (fabs(z[j]-z[k]) < aReg) {
      d = sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
      uReg[i][j][k] = qp*qp*lB1    * tanh(d/aReg)/d; continue;
    }
    uReg[i][j][k] = u[i][j][k];
  }}}
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (fabs(z[j]-z[k]) < aReg) {
      d = sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
      uReg[i][j][k+Nz] = -qp*qm*lB1    * tanh(d/aReg)/d; continue;
    }
    uReg[i][j][k+Nz] = u[i][j][k+Nz];
    uReg[i][j+Nz][k] = uReg[i][j][k+Nz];
  }}}
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (fabs(z[j]-z[k]) < aReg) {
      d = sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
      uReg[i][j+Nz][k+Nz] = qm*qm*lB1    * tanh(d/aReg)/d; continue;
    }
    uReg[i][j+Nz][k+Nz] = u[i][j+Nz][k+Nz];
  }}}
*/
  /* The DHT of the alternative regularized Coulomb potential between two point charges (JZ) */
/*
  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
      for (i=0;i<Nr;i++) { dc[i] = uReg[i][j][k]; }
      gsl_dht_apply(t, dc, dcHT);
      for (i=0;i<Nr;i++) { uRegHT[i][j][k] = dcHT[i];}
  }}
*/
}

void CoulombFluid::SetPairPotential2DielectricWalls() {

  /* Dielectric interfaces are located at z[0]-ap and at z[Nz-1]+ap */

  double Vdepthp, Vdepthm, A13, A12, *norm, *VexIm, B, *kFT2, Nz2=8000;

  norm = (double*) calloc (Nr, sizeof(double));
  VexIm = (double*) calloc (Nz, sizeof(double));

  kFT2 = (double*) calloc (Nz2, sizeof(double));
  for(k=0;k<Nz2;k++) kFT2[k] = k*0.05;


  Vlength = 5.0/kappa1;//lB1*5.0;
  Vdepthp = -depth;
  Vdepthm = -depth;

  Vlength = 2.1*(z[1]-z[0]);

  A13 = (e1-e3)/(e1+e3); A12 = (e1-e2)/(e1+e2);

  for (i=0;i<Nr;i++){
    norm[i] = 1.0 - A13*A12*exp(-2.0*Htot*kFT[i]);
    if (norm[i]>0) norm[i] = 1.0/norm[i];
    else norm[i] = 1.0/(-2.0*Htot*kFT[i]);
  }

  //sigma = pow(5000,-1.0)*1e2;

  aReg = 5.0/kFT[Nz-1];
  cout << "highest value for kFT is " << kFT[Nz-1] << "/nm, and aReg = " << aReg << " nm" << endl;

  /* External field */
  for (j=0;j<Nz;j++) {
    //Vextp[j] = Vdepthp*exp(-(z[j]-z[0])/Vlength) + Vdepthp*exp((z[j]-z[Nz-1])/Vlength);
    //Vextm[j] = Vdepthm*exp(-(z[j]-z[0])/Vlength) + Vdepthm*exp((z[j]-z[Nz-1])/Vlength);
    //    if ( (z[j]-z[0]) < 2*ap or (z[Nz-1]-z[j])< 2*ap) { Vextp[j] = Vdepthp;    Vextm[j] = Vdepthm; } else Vextp[j] = Vextm[j] = 0;
    //    Vextp[j] = 0.5*lB1*z[j]*sigma; Vextm[j] = -0.5*lB1*z[j]*sigma;

    //Vextp[j] = (2*z[j]-z[Nz-1]-z[0])/(z[Nz-1]-z[0])*qpv[j]*sigma;
    Vextp[j] =  qpv[j]*sigma *z[j]*kappa1/H;
    Vextm[j] = -qmv[j]/qpv[j]*Vextp[j];
    Vextp[j] =  qpv[j]*Vtot / 2.0 * e/(kB*T) * z[j] * 2.0/Htot + qpv[j]*QDNA[j];    //fixed potential
    Vextm[j] = -qmv[j]*Vtot / 2.0 * e/(kB*T) * z[j] * 2.0/Htot - qmv[j]*QDNA[j];    //fixed potential
    phi[j]   = -Vextp[j] / qpv[j];
    //if ( j < Nl or (Nz-Nl-1)<j ) { Vextp[j] += Vdepthp;    Vextm[j] += Vdepthm; };
    if ( z[j]-z[0] < Lw) { Vextp[j] += Vdepthp*((-z[j]+z[0]+Lw)/Lw);    Vextm[j] += Vdepthm*((-z[j]+z[0]+Lw)/Lw); };
    if ( z[Nz-1]-z[j] < Lw) { Vextp[j] += -Vdepthp*((z[Nz-1]-z[j]-Lw)/Lw);    Vextm[j] += -Vdepthm*((z[Nz-1]-z[j]-Lw)/Lw); };
  }

  //Image Self-Interaction (tested)
  for (j=0;j<Nz;j++) {
    VexIm[j] = lB1*qp*qp/Htot*log(1.0/(1.0-A13*A12));
    //for(k=1;k<Nz2;k++) VexIm[j] += lB1*qp*qp * (kFT2[k]-kFT2[k-1]) * pow(1.0 - A13*A12*exp(-2.0*Htot*kFT2[k]),-1.0) * (A13*exp(-2*(z[Nz-1]-z[j]+ GridCutoff*ap/2.0)*kFT2[k]) + A12*exp(-2*(z[j]-z[0]+ GridCutoff*ap/2.0)*kFT2[k]));
    for(k=1;k<Nr;k++) VexIm[j] += lB1*qp*qp * (kFT[k]-kFT[k-1]) * pow(1.0-A13*A12*exp(-2*Htot*kFT[k]),-1.0) * (A13*exp(-2*(z[Nz-1]-z[j]+ GridCutoff*ap/2.0)*kFT[k]) + A12*exp(-2*(z[j]-z[0]+ GridCutoff*ap/2.0)*kFT[k]));
    //for(k=1;k<Nr;k++) VexIm[j] += lB1*qp*qp * (kFT[k]-kFT[k-1]) * pow(1.0-A13*A12*exp(-2*(z[Nz-1]-z[0]+2*ap)*kFT[k]), -1.0) * (A13*exp(-2*(z[Nz-1]-z[j]+ap)*kFT[k])+A12*exp(-2*(z[j]-z[0]+ap)*kFT[k]));
    Vextp[j] += VexIm[j];
```

```
  Vextm[j] += VexIm[j]*qm*qm/qp/qp;
}

//Lennard-Jones
//cout << "Lennard-Jones interaction with wall included" << endl;
//for (j=0;j<Nz;j++) {
//  B = pow(ap/(z[j]+1.5),6.0);   if (B > 0.5) { B = 4.0*(B*B-B)+1.0; Vextp[j] += B; Vextm[j] += B; }
//  B = pow(ap/(z[j]-1.5),6.0);   if (B > 0.5) { B = 4.0*(B*B-B)+1.0; Vextp[j] += B; Vextm[j] += B; }
//}

/* The Coulomb potential between two point charges */
/* Positive particles are counted from j,k = 0 to N - 1; negative particles from j,k = N to 2N - 1. */
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
  if (j==k and r[i]==0) {
    u[0][j][k] = Large; continue;
  }
  u[i][j][k] = qp*qp*lB1/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
}}}
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
  if (j==k and r[i]==0) {
    u[i][j+Nz][k] = -Large; continue;
    u[i][j][k+Nz] = u[i][j+Nz][k];
  }
  u[i][j+Nz][k] = -qp*qm*lB1/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
  u[i][j][k+Nz] = u[i][j+Nz][k];
}}}
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
  if (j==k and r[i]==0) {
    u[i][j+Nz][k+Nz] = Large; continue;
  }
  u[i][j+Nz][k+Nz] = qm*qm*lB1/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
}}}

/* The DHT of the Coulomb potential between two point charges */
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
  if (r[i] == 0.0) {uHT[
  uHT[i][j][k] =  2*pi*qp*qp*lB1/kFT[i]*exp(-fabs(z[j]-z[k])*kFT[i]);
  uHT[i][j][k] += 2*pi*qp*qp*lB1/kFT[i]*norm[i]*(A12*exp(-(z[j]+z[k]-2*z[0]+ GridCutoff*ap)*kFT[i]) + A13*exp(-(2*z[Nz-1]+ GridCutoff*ap - z[j]-z[k])*kFT[i]));
  uHT[i][j][k] += 2*pi*qp*qp*lB1/kFT[i]*norm[i]* A13*A12*(exp(-2.0*(Htot + 0.5*(z[j]-z[k]))*kFT[i])+exp(-2.0*(Htot - 0.5*(z[j]-z[k]))*kFT[i]));
}}}
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=Nz;k<2*Nz;k++){
  if (r[i] == 0.0) {uHT[i][j][k] = -Large; uHT[i][j+Nz][k-Nz] = -Large; continue;}
  uHT[i][j][k] =  -2*pi*qp*qm*lB1/kFT[i]*exp(-fabs(z[j]-z[k-Nz])*kFT[i]);
  uHT[i][j][k] += -2*pi*qp*qm*lB1/kFT[i]*norm[i]*(A12*exp(-(z[j]+z[k-Nz]-2*z[0]+ GridCutoff*ap)*kFT[i]) + A13*exp(-(2*z[Nz-1]+ GridCutoff*ap - z[j]-z[k-Nz])*kFT[i]));
  uHT[i][j][k] += -2*pi*qp*qm*lB1/kFT[i]*norm[i]* A13*A12*(exp(-2.0*(Htot + 0.5*(z[j]-z[k-Nz]))*kFT[i])+exp(-2.0*(Htot - 0.5*(z[j]-z[k-Nz]))*kFT[i]));
  uHT[i][j+Nz][k-Nz] = uHT[i][j][k];
}}}
for (i=0;i<Nr;i++){    for (j=Nz;j<2*Nz;j++){      for (k=Nz;k<2*Nz;k++){
  if (r[i] == 0.0) {uHT[i][j][k] = Large; continue;}
  uHT[i][j][k] =  2*pi*qm*qm*lB1/kFT[i]*exp(-fabs(z[j-Nz]-z[k-Nz])*kFT[i]);
  uHT[i][j][k] += 2*pi*qm*qm*lB1/kFT[i]*norm[i]*(A12*exp(-(z[j-Nz]+z[k-Nz]-2*z[0]+ GridCutoff*ap)*kFT[i]) + A13*exp(-(2*z[Nz-1]+ GridCutoff*ap - z[j-Nz]-z[k-Nz])*kFT[i]));
  uHT[i][j][k] += 2*pi*qm*qm*lB1/kFT[i]*norm[i]* A13*A12*(exp(-2.0*(Htot + 0.5*(z[j-Nz]-z[k-Nz]))*kFT[i])+exp(-2.0*(Htot - 0.5*(z[j-Nz]-z[k-Nz]))*kFT[i]));
}}}

/* The regularized Coulomb potential between two point charges */

for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
  if (fabs(z[j]-z[k]) < aReg) {
    uReg[i][j][k] = qp*qp*lB1/sqrt(r[i]*r[i] + aReg*aReg); continue;
  }
  uReg[i][j][k] = u[i][j][k];
}}}
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
  if (fabs(z[j]-z[k]) < aReg) {
    uReg[i][j][k+Nz] = -qp*qm*lB1/sqrt(r[i]*r[i] + aReg*aReg);
    uReg[i][j+Nz][k] = uReg[i][j][k+Nz];      continue;
  }
  uReg[i][j][k+Nz] = u[i][j][k+Nz];
  uReg[i][j+Nz][k] = uReg[i][j][k+Nz];
}}}
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
  if (fabs(z[j]-z[k]) < aReg) {
    uReg[i][j+Nz][k+Nz] = qm*qm*lB1/sqrt(r[i]*r[i] + aReg*aReg); continue;
  }
  uReg[i][j+Nz][k+Nz] = u[i][j+Nz][k+Nz];
}}}

/* The DHT of the regularized Coulomb potential between two point charges */
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
  if (r[i] == 0.0) {uRegHT[i][j][k] = Large; continue;}
  uRegHT[i][j][k] = uHT[i][j][k];
  if (fabs(z[j]-z[k]) < aReg) {
    uRegHT[i][j][k] += 2*pi*qp*qp*lB1/kFT[i]*( exp(-aReg*kFT[i]) - exp(-fabs(z[j]-z[k])*kFT[i]) );
  }
  if (fabs(z[j]+z[k]-2*z[0]+ GridCutoff*ap) < aReg) {
    uRegHT[i][j][k] += 2*pi*qp*qp*lB1/kFT[i]*norm[i]*A12*( exp(-aReg*kFT[i]) - exp(-(z[j]+z[k]-2*z[0]+ GridCutoff*ap)*kFT[i]));
  }
  if (fabs(2*z[Nz-1]+ GridCutoff*ap-z[j]-z[k]) < aReg) {
    uRegHT[i][j][k] += 2*pi*qp*qp*lB1/kFT[i]*norm[i]*A13*( exp(-aReg*kFT[i]) - exp(-(2*z[Nz-1]+ GridCutoff*ap-z[j]-z[k])*kFT[i]));
  }
}}}
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=Nz;k<2*Nz;k++){
  if (r[i] == 0.0) {uRegHT[i][j][k] = -Large; continue;}
  uRegHT[i][j][k] = uHT[i][j][k];
  if (fabs(z[j]-z[k-Nz]) < aReg) {
    uRegHT[i][j][k] += -2*pi*qp*qm*lB1/kFT[i]*( exp(-aReg*kFT[i]) - exp(-fabs(z[j]-z[k-Nz])*kFT[i]) );
  }
  if (fabs(z[j]+z[k-Nz]-2*z[0]+ GridCutoff*ap) < aReg) {
    uRegHT[i][j][k] += -2*pi*qp*qm*lB1/kFT[i]*norm[i]*A12*( exp(-aReg*kFT[i]) - exp(-(z[j]+z[k-Nz]-2*z[0]+ GridCutoff*ap)*kFT[i]));
  }
  if (fabs(2*z[Nz-1]+ GridCutoff*ap-z[j]-z[k]) < aReg) {
    uRegHT[i][j][k] += -2*pi*qp*qm*lB1/kFT[i]*norm[i]*A13*( exp(-aReg*kFT[i]) - exp(-(2*z[Nz-1]+ GridCutoff*ap-z[j]-z[k-Nz])*kFT[i]));
  }
  uRegHT[i][j+Nz][k-Nz] = uRegHT[i][j][k];
}}}
for (i=0;i<Nr;i++){    for (j=Nz;j<2*Nz;j++){      for (k=Nz;k<2*Nz;k++){
```

```
    if (r[i] == 0.0) {uRegHT[i][j][k] = Large; continue;}
    uRegHT[i][j][k] = uHT[i][j][k];
    if (fabs(z[j-Nz]-z[k-Nz]) < aReg) {
      uRegHT[i][j][k] += 2*pi*qm*qm*lB1/kFT[i]*( exp(-aReg*kFT[i]) - exp(-fabs(z[j-Nz]-z[k-Nz])*kFT[i]) );
    }
    if (fabs(z[j]+z[k]-2*z[0]+ GridCutoff*ap) < aReg) {
      uRegHT[i][j][k] += 2*pi*qm*qm*lB1/kFT[i]*norm[i]*A12*( exp(-aReg*kFT[i]) - exp(-(z[j-Nz]+z[k-Nz]-2*z[0]+ GridCutoff*ap)*kFT[i]));
    }
    if (fabs(2*z[Nz-1]+ GridCutoff*ap-z[j]-z[k]) < aReg) {
      uRegHT[i][j][k] += 2*pi*qm*qm*lB1/kFT[i]*norm[i]*A13*( exp(-aReg*kFT[i]) - exp(-(2*z[Nz-1]+ GridCutoff*ap-z[j-Nz]-z[k-Nz])*kFT[i]));
    }
  }}}
  /* The alternative regularized Coulomb potential between two point charges, choice of author (JZ) */
  /*
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (fabs(z[j]-z[k]) < aReg) {
      d = sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
      uReg[i][j][k] = qp*qp*lB1    * tanh(d/aReg)/d; continue;
    }
    uReg[i][j][k] = u[i][j][k];
  }}}
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (fabs(z[j]-z[k]) < aReg) {
      d = sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
      uReg[i][j][k+Nz] = -qp*qm*lB1    * tanh(d/aReg)/d; continue;
    }
    uReg[i][j][k+Nz] = u[i][j][k+Nz];
    uReg[i][j+Nz][k] = uReg[i][j][k+Nz];
  }}}
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (fabs(z[j]-z[k]) < aReg) {
      d = sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
      uReg[i][j+Nz][k+Nz] = qm*qm*lB1    * tanh(d/aReg)/d; continue;
    }
    uReg[i][j+Nz][k+Nz] = u[i][j+Nz][k+Nz];
  }}}
  */
  /* The DHT of the alternative regularized Coulomb potential between two point charges (JZ) */
  /*
  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
      for (i=0;i<Nr;i++) { dc[i] = uReg[i][j][k]; }
      gsl_dht_apply(t, dc, dcHT);
      for (i=0;i<Nr;i++) { uRegHT[i][j][k] = dcHT[i];}
  }}
  */
  free(norm); free(VexIm); free(kFT2);
}

void CoulombFluid::SetRegularizedPairPotential2DielectricWalls() {

  /* Dielectric interfaces are located at z[0]-ap and at z[Nz-1]+ap */

  double Vdepthp, Vdepthm, A13, A12, *norm, *Reg, *VexIm, B, *kFT2, Nz2=8000;

  norm = (double*) calloc (Nr, sizeof(double));
  Reg = (double*) calloc (Nr, sizeof(double));
  VexIm = (double*) calloc (Nz, sizeof(double));

  kFT2 = (double*) calloc (Nz2, sizeof(double));
  for(k=0;k<Nz2;k++) kFT2[k] = k*0.05;


  Vlength = 5.0/kappa1;//lB1*5.0;
  Vdepthp = -depth;
  Vdepthm = -depth;

  Vlength = 2.1*(z[1]-z[0]);

  A13 = (e1-e3)/(e1+e3); A12 = (e1-e2)/(e1+e2);

  for (i=0;i<Nr;i++){
    norm[i] = 1.0 - A13*A12*exp(-2.0*Htot*kFT[i]);
    if (norm[i]>0) norm[i] = 1.0/norm[i];
    else norm[i] = 1.0/(-2.0*Htot*kFT[i]);
  }


  for (i=0;i<Nr;i++) Reg[i] = exp(-2.0*r[i]*r[i]/r[Nr-1]/r[Nr-1]);


  //sigma = pow(5000,-1.0)*1e2;

  aReg = 5.0/kFT[Nz-1];
  cout << "highest value for kFT is " << kFT[Nz-1] << "/nm, and aReg = " << aReg << " nm" << endl;

  /* External field */
  for (j=0;j<Nz;j++) {
    //Vextp[j] = Vdepthp*exp(-(z[j]-z[0])/Vlength) + Vdepthp*exp((z[j]-z[Nz-1])/Vlength);
    //Vextm[j] = Vdepthm*exp(-(z[j]-z[0])/Vlength) + Vdepthm*exp((z[j]-z[Nz-1])/Vlength);
    //    if ( (z[j]-z[0]) < 2*ap or (z[Nz-1]-z[j])< 2*ap) { Vextp[j] = Vdepthp;    Vextm[j] = Vdepthm; } else Vextp[j] = Vextm[j] = 0;
    //    Vextp[j] = 0.5*lB1*z[j]*sigma; Vextm[j] = -0.5*lB1*z[j]*sigma;

    //Vextp[j] = (2*z[j]-z[Nz-1]-z[0])/(z[Nz-1]-z[0])*qpv[j]*sigma;
    Vextp[j] =  qpv[j]*sigma *z[j]*kappa1/H;
    Vextm[j] = -qmv[j]/qpv[j]*Vextp[j];
    Vextp[j] =  qpv[j]*Vtot / 2.0 * e/(kB*T) * z[j] * 2.0/Htot + qpv[j]*QDNA[j];    //fixed potential
    Vextm[j] = -qmv[j]*Vtot / 2.0 * e/(kB*T) * z[j] * 2.0/Htot - qmv[j]*QDNA[j];    //fixed potential
    phi[j]   = -Vextp[j] / qpv[j];
    //if ( j < Nl or (Nz-Nl-1)<j ) { Vextp[j] += Vdepthp;    Vextm[j] += Vdepthm; };
    if ( z[j]-z[0] < Lw) { Vextp[j] += Vdepthp*((-z[j]+z[0]+Lw)/Lw);    Vextm[j] += Vdepthm*((-z[j]+z[0]+Lw)/Lw); };
    if ( z[Nz-1]-z[j] < Lw) { Vextp[j] += -Vdepthp*((z[Nz-1]-z[j]-Lw)/Lw);    Vextm[j] += -Vdepthm*((z[Nz-1]-z[j]-Lw)/Lw); };
  }

  //Image Self-Interaction (tested)
  for (j=0;j<Nz;j++) {
```

```
  VexIm[j] = lB1*qp*qp/Htot*log(1.0/(1.0-A13*A12));
  //for(k=1;k<Nz2;k++) VexIm[j] += lB1*qp*qp * (kFT2[k]-kFT2[k-1]) * pow(1.0 - A13*A12*exp(-2.0*Htot*kFT2[k]),-1.0) * (A13*exp(-2*(z[Nz-1]-z[j]+ GridCutoff*ap/2.0)*kFT2[k]) + A12*exp(-2*(z[j]-z[0]+ GridCutoff*ap/2.0)*kFT2[k]));
  for(k=1;k<Nr;k++) VexIm[j] += lB1*qp*qp * (kFT[k]-kFT[k-1]) * pow(1.0-A13*A12*exp(-2*Htot*kFT[k]),-1.0) * (A13*exp(-2*(z[Nz-1]-z[j]+ GridCutoff*ap/2.0)*kFT[k]) + A12*exp(-2*(z[j]-z[0]+ GridCutoff*ap/2.0)*kFT[k]));
  //for(k=1;k<Nr;k++) VexIm[j] += lB1*qp*qp * (kFT[k]-kFT[k-1]) * pow(1.0-A13*A12*exp(-2*(z[Nz-1]-z[0]+2*ap)*kFT[k]), -1.0) * (A13*exp(-2*(z[Nz-1]-z[j]+ap)*kFT[k])+A12*exp(-2*(z[j]-z[0]+ap)*kFT[k]));
  Vextp[j] += VexIm[j];
  Vextm[j] += VexIm[j]*qm*qm/qp/qp;
}

//Lennard-Jones
cout << "Lennard-Jones interaction with wall included" << endl;
for (j=0;j<Nz;j++) {
  B = pow(ap/(z[j]+1.5),6.0);     if (B > 0.5) { B = 4.0*(B*B-B)+1.0; Vextp[j] += B; Vextm[j] += B; }
  B = pow(ap/(z[j]-1.5),6.0);     if (B > 0.5) { B = 4.0*(B*B-B)+1.0; Vextp[j] += B; Vextm[j] += B; }
}

/* The Coulomb potential between two point charges */
/* Positive particles are counted from j,k = 0 to N - 1; negative particles from j,k = N to 2N - 1. */
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
  if (j==k and r[i]==0) {
    u[0][j][k] = Large; continue;
  }
  u[i][j][k] = qp*qp*lB1/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)) * Reg[i];
}}}
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
  if (j==k and r[i]==0) {
    u[i][j+Nz][k] = -Large; continue;
    u[i][j][k+Nz] = u[i][j+Nz][k];
  }
  u[i][j+Nz][k] = -qp*qm*lB1/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)) * Reg[i];
  u[i][j][k+Nz] = u[i][j+Nz][k];
}}}
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
  if (j==k and r[i]==0) {
    u[i][j+Nz][k+Nz] = Large; continue;
  }
  u[i][j+Nz][k+Nz] = qm*qm*lB1/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)) * Reg[i];
}}}

/* The DHT of the Coulomb potential between two point charges */
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
  if (r[i] == 0.0) {uHT[i][j][k] = Large; continue;}
  uHT[i][j][k] =  2*pi*qp*qp*lB1/kFT[i]*exp(-fabs(z[j]-z[k])*kFT[i]);
  uHT[i][j][k] += 2*pi*qp*qp*lB1/kFT[i]*norm[i]*(A12*exp(-(z[j]+z[k]-2*z[0]+ GridCutoff*ap)*kFT[i]) + A13*exp(-(2*z[Nz-1]+ GridCutoff*ap - z[j]-z[k])*kFT[i]));
  uHT[i][j][k] += 2*pi*qp*qp*lB1/kFT[i]*norm[i]* A13*A12*(exp(-2.0*(Htot + 0.5*(z[j]-z[k]))*kFT[i])+exp(-2.0*(Htot - 0.5*(z[j]-z[k]))*kFT[i]));
}}}
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=Nz;k<2*Nz;k++){
  if (r[i] == 0.0) {uHT[i][j][k] = -Large; uHT[i][j+Nz][k-Nz] = -Large; continue;}
  uHT[i][j][k] =  -2*pi*qp*qm*lB1/kFT[i]*exp(-fabs(z[j]-z[k-Nz])*kFT[i]);
  uHT[i][j][k] += -2*pi*qp*qm*lB1/kFT[i]*norm[i]*(A12*exp(-(z[j]+z[k-Nz]-2*z[0]+ GridCutoff*ap)*kFT[i]) + A13*exp(-(2*z[Nz-1]+ GridCutoff*ap - z[j]-z[k-Nz])*kFT[i]));
  uHT[i][j][k] += -2*pi*qp*qm*lB1/kFT[i]*norm[i]* A13*A12*(exp(-2.0*(Htot + 0.5*(z[j]-z[k-Nz]))*kFT[i])+exp(-2.0*(Htot - 0.5*(z[j]-z[k-Nz]))*kFT[i]));
  uHT[i][j+Nz][k-Nz] = uHT[i][j][k];
}}}
for (i=0;i<Nr;i++){    for (j=Nz;j<2*Nz;j++){      for (k=Nz;k<2*Nz;k++){
  if (r[i] == 0.0) {uHT[i][j][k] = Large; continue;}
  uHT[i][j][k] =  2*pi*qm*qm*lB1/kFT[i]*exp(-fabs(z[j-Nz]-z[k-Nz])*kFT[i]);
  uHT[i][j][k] += 2*pi*qm*qm*lB1/kFT[i]*norm[i]*(A12*exp(-(z[j-Nz]+z[k-Nz]-2*z[0]+ GridCutoff*ap)*kFT[i]) + A13*exp(-(2*z[Nz-1]+ GridCutoff*ap - z[j-Nz]-z[k-Nz])*kFT[i]));
  uHT[i][j][k] += 2*pi*qm*qm*lB1/kFT[i]*norm[i]* A13*A12*(exp(-2.0*(Htot + 0.5*(z[j-Nz]-z[k-Nz]))*kFT[i])+exp(-2.0*(Htot - 0.5*(z[j-Nz]-z[k-Nz]))*kFT[i]));
}}}
/* The regularized Coulomb potential between two point charges */

for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
  if (fabs(z[j]-z[k]) < aReg) {
    uReg[i][j][k] = qp*qp*lB1/sqrt(r[i]*r[i] + aReg*aReg) * Reg[i]; continue;
  }
  uReg[i][j][k] = u[i][j][k];
}}}
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
  if (fabs(z[j]-z[k]) < aReg) {
    uReg[i][j][k+Nz] = -qp*qm*lB1/sqrt(r[i]*r[i] + aReg*aReg) * Reg[i];
    uReg[i][j+Nz][k] = uReg[i][j][k+Nz];      continue;
  }
  uReg[i][j][k+Nz] = u[i][j][k+Nz];
  uReg[i][j+Nz][k] = uReg[i][j][k+Nz];
}}}
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
  if (fabs(z[j]-z[k]) < aReg) {
    uReg[i][j+Nz][k+Nz] = qm*qm*lB1/sqrt(r[i]*r[i] + aReg*aReg) * Reg[i]; continue;
  }
  uReg[i][j+Nz][k+Nz] = u[i][j+Nz][k+Nz];
}}}

/* The DHT of the regularized Coulomb potential between two point charges */
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
  if (r[i] == 0.0) {uRegHT[i][j][k] = Large; continue;}
  uRegHT[i][j][k] = uHT[i][j][k];
  if (fabs(z[j]-z[k]) < aReg) {
    uRegHT[i][j][k] += 2*pi*qp*qp*lB1/kFT[i]*( exp(-aReg*kFT[i]) - exp(-fabs(z[j]-z[k])*kFT[i]) );
  }
  if (fabs(z[j]+z[k]-2*z[0]+ GridCutoff*ap) < aReg) {
    uRegHT[i][j][k] += 2*pi*qp*qp*lB1/kFT[i]*norm[i]*A12*( exp(-aReg*kFT[i]) - exp(-(z[j]+z[k]-2*z[0]+ GridCutoff*ap)*kFT[i]));
  }
  if (fabs(2*z[Nz-1]+ GridCutoff*ap-z[j]-z[k]) < aReg) {
    uRegHT[i][j][k] += 2*pi*qp*qp*lB1/kFT[i]*norm[i]*A13*( exp(-aReg*kFT[i]) - exp(-(2*z[Nz-1]+ GridCutoff*ap-z[j]-z[k])*kFT[i]));
  }
}}}
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=Nz;k<2*Nz;k++){
  if (r[i] == 0.0) {uRegHT[i][j][k] = -Large; continue;}
  uRegHT[i][j][k] = uHT[i][j][k];
  if (fabs(z[j]-z[k-Nz]) < aReg) {
    uRegHT[i][j][k] += -2*pi*qp*qm*lB1/kFT[i]*( exp(-aReg*kFT[i]) - exp(-fabs(z[j]-z[k-Nz])*kFT[i]) );
  }
  if (fabs(z[j]+z[k-Nz]-2*z[0]+ GridCutoff*ap) < aReg) {
    uRegHT[i][j][k] += -2*pi*qp*qm*lB1/kFT[i]*norm[i]*A12*( exp(-aReg*kFT[i]) - exp(-(z[j]+z[k-Nz]-2*z[0]+ GridCutoff*ap)*kFT[i]));
  }
  if (fabs(2*z[Nz-1]+ GridCutoff*ap-z[j]-z[k]) < aReg) {
```

```
      uRegHT[i][j][k] += -2*pi*qp*qm*lB1/kFT[i]*norm[i]*A13*( exp(-aReg*kFT[i]) - exp(-(2*z[Nz-1]+ GridCutoff*ap-z[j]-z[k-Nz])*kFT[i]));
    }
    uRegHT[i][j+Nz][k-Nz] = uRegHT[i][j][k];
  }}}
 for (i=0;i<Nr;i++){    for (j=Nz;j<2*Nz;j++){      for (k=Nz;k<2*Nz;k++){
    if (r[i] == 0.0) {uRegHT[i][j][k] = Large; continue;}
    uRegHT[i][j][k] = uHT[i][j][k];
    if (fabs(z[j-Nz]-z[k-Nz]) < aReg) {
      uRegHT[i][j][k] += 2*pi*qm*qm*lB1/kFT[i]*( exp(-aReg*kFT[i]) - exp(-fabs(z[j-Nz]-z[k-Nz])*kFT[i]) );
    }
    if (fabs(z[j]+z[k]-2*z[0]+ GridCutoff*ap) < aReg) {
      uRegHT[i][j][k] += 2*pi*qm*qm*lB1/kFT[i]*norm[i]*A12*( exp(-aReg*kFT[i]) - exp(-(z[j-Nz]+z[k-Nz]-2*z[0]+ GridCutoff*ap)*kFT[i]));
    }
    if (fabs(2*z[Nz-1]+ GridCutoff*ap-z[j]-z[k]) < aReg) {
      uRegHT[i][j][k] += 2*pi*qm*qm*lB1/kFT[i]*norm[i]*A13*( exp(-aReg*kFT[i]) - exp(-(2*z[Nz-1]+ GridCutoff*ap-z[j-Nz]-z[k-Nz])*kFT[i]));
    }
  }}}

  free(norm); free(Reg); free(VexIm); free(kFT2);
}

void CoulombFluid::GnuplotDensity(){

  vector<double> z1,n1,cp1,cm1;

  g1.remove_tmpfiles();

  try
  {

    g1.reset_plot();
    //cout << endl << endl << "plotting the density profiles" << endl;
    g1.unset_grid();
    g1.set_style("lines");//.set_yrange(0.0,2.0*cs1);//.set_xrange(-1.2 * z[1],1.2*z[Nz-2]).set_yrange(0.0,2.0*cs1);

    for (j=1;j<Nz-1;j++) {      z1.push_back(z[j]);        n1.push_back(n[j]);        cp1.push_back(cp[j]/NA);        cm1.push_back(cm[j]/NA);      }
    g1.plot_xy(z1,cp1,"cation density (M)");
    g1.plot_xy(z1,cm1,"anion density (M)");
    //g1.plot_xy(z1,n1,"areal density per slab");

    z1.clear();      n1.clear();      cp1.clear();      cm1.clear();

  }
  catch (GnuplotException ge)
  {
    cout << ge.what() << endl;
  }

}

void CoulombFluid::GnuplotCorrelationFunctions(){

  vector<double> z1,r1,h1;

  for (i=0;i<Nr;i++) { for (j=0;j<2*Nz;j++) {for (k=0;k<2*Nz;k++) {
    if (Qs[i][j][k]!=0.0) h[i][j][k] = -1.0 + 1e-10;
    //if (Qs[i][j][k]!=0.0) Qs[i][j][k] -= 2.0;
  }}}

  try
  {

    if(iterations%1 == 0) {g2.remove_tmpfiles();}
    g2.reset_plot();
    cout << endl << endl << "plotting the density profiles" << endl;
    g2.unset_grid();
    g2.unset_surface();
    g2.set_hidden3d();
    //g2.set_contour("surface");
    //g2.set_style("lines");
    g2.cmd("set dgrid3d 50,50,1");
    g2.cmd("set pm3d");
    g2.cmd("set zrange [0.01:1]");
    g2.cmd("set border linewidth 2.0");
    //g2.cmd("set zrange[0:*]");
    g2.set_zlogscale(10);


    for (i=0;i<Nr;i++) {  for (j=0;j<Nz;j++){
      r1.push_back(-r[i]);
      z1.push_back(z[j]);
      h1.push_back(fabs(h[i][j][Nz]));
    }
    }
    for (i=0;i<Nr;i++) {  for (j=0;j<Nz;j++){
      r1.push_back(r[i]);
      z1.push_back(z[j]);
      h1.push_back(fabs(h[i][j][Nz]));
    }
    }

    g2.plot_xyz(z1,r1,h1,"|h+-|");

    z1.clear();      r1.clear();      h1.clear();

  }
  catch (GnuplotException ge)
  {
    cout << ge.what() << endl;
  }

}

void CoulombFluid::InitiateCorrelationFunctions(double *hp,double *hm,double *hpm,double *dcpp,double *dcmm,double *dcpm,double *s1,int N,int medium) {
```

```
  double dist,frac;
  int l;

  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    dist = sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
    l=0; while (s1[l]<dist and l<N) l++;
    frac = (dist-s1[l-1])/(s1[l]-s1[l-1]);
    if (pow(-1.0,1.0*medium)*z[j] > 0.0) {
      h[i][j][k] = hp[l] + frac*(hp[l]-hp[l-1]);
      h[i][j+Nz][k+Nz] = hm[l] + frac*(hm[l]-hm[l-1]);
      h[i][j][k+Nz] = h[i][j+Nz][k] = hpm[l] + frac*(hpm[l]-hpm[l-1]);
      c[i][j][k] = dcpp[l] + frac*(dcpp[l]-dcpp[l-1]);
      c[i][j+Nz][k+Nz] = dcmm[l] + frac*(dcmm[l]-dcmm[l-1]);
      c[i][j][k+Nz] = c[i][j+Nz][k] = dcpm[l] + frac*(dcpm[l]-dcpm[l-1]);
    }
    if (l >= N) { h[i][j][k] = 0.0; c[i][j][k] = 0.0; }
    y[i][j][k] = h[i][j][k]-c[i][j][k];
  }}}
/*
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<j;k++){
    if (pow(-1.0,1.0*medium)*z[j] > 0.0) {
      h[i][j][k] = h[i][k][j];
      c[i][j][k] = c[i][k][j];
    }
  }}}
*/
/*
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (r[i]*r[i] + pow(z[j]-z[k],2.0) < 4*ap*ap) continue;
    newc[i][j][k] = - u[i][j][k]+h[i][j][k] - log(h[i][j][k] + 1.0);
  }}}
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=Nz;k<2*Nz;k++){
    if (r[i]*r[i] + pow(z[j]-z[k-Nz],2.0) < pow(ap+am,2.0)) continue;
    newc[i][j][k] = - u[i][j][k]+h[i][j][k] - log(h[i][j][k] + 1.0);
    newc[i][j+Nz][k-Nz] = newc[i][j][k];
  }}}
  for (i=0;i<Nr;i++){    for (j=Nz;j<2*Nz;j++){      for (k=Nz;k<2*Nz;k++){
    if (r[i]*r[i] + pow(z[j-Nz]-z[k-Nz],2.0) < 4*am*am) continue;
    newc[i][j][k] = - u[i][j][k]+ h[i][j][k] - log(h[i][j][k] + 1.0);
  }}}
*/

}

void CoulombFluid::InitiateCorrelationFunctions2Walls(double *hp,double *hm,double *hpm,double *dcpp,double *dcmm,double *dcpm,double *s1,int N,int medium) {

  double dist,frac;
  int l;

  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    dist = sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
    l=0; while (s1[l]<dist and l<N) l++;
    frac = (dist-s1[l-1])/(s1[l]-s1[l-1]);
    h[i][j][k] = hp[l] + frac*(hp[l]-hp[l-1]);
    h[i][j+Nz][k+Nz] = hm[l] + frac*(hm[l]-hm[l-1]);
    h[i][j][k+Nz] = h[i][j+Nz][k] = hpm[l] + frac*(hpm[l]-hpm[l-1]);
    if (dist < 4*ap*ap) {
      h[i][j][k] = -1.0;
      if(s1[l] > 2*ap)    h[i][j][k] = hp[l+1] + frac*(hp[l+1]-hp[l]);
    }
    if (dist < 4*am*am) {
      h[i][j][k] = -1.0;
      if(s1[l] > 2*am)    h[i][j+Nz][k+Nz] = hm[l+1] + frac*(hm[l+1]-hm[l]);
    }
    if (dist < pow(ap+am,2.0)) {
      h[i][j][k] = -1.0;
      if(s1[l] > ap+am)    h[i][j][k+Nz] = h[i][j+Nz][k] = hpm[l+1] + frac*(hpm[l+1]-hpm[l]);
    }
    c[i][j][k] = dcpp[l] + frac*(dcpp[l]-dcpp[l-1]);
    c[i][j+Nz][k+Nz] = dcmm[l] + frac*(dcmm[l]-dcmm[l-1]);
    c[i][j][k+Nz] = c[i][j+Nz][k] = dcpm[l] + frac*(dcpm[l]-dcpm[l-1]);
    if (l >= N) { h[i][j][k] = 0.0; c[i][j][k] = 0.0; }
    if (dist == 0.0) {c[i][j][k] = dcpp[i+1]-(dcpp[i+2]-dcpp[i+1])*(r[i+1]-r[i])/(r[i+2]-r[i+1]);}
    y[i][j][k] = h[i][j][k] - c[i][j][k];
  }}}

  for (i=0;i<Nr;i++){      for (k=0;k<2*Nz;k++){
      htest[i][k]=h[i][3*Nz/2][k];
  }}

}

void CoulombFluid::SetHankelTransformPairPotential() {

  double image, screen;

  screen = 2*lB1*lB2/(lB2+lB1);
  image = (lB2-lB1)/(lB2+lB1);

   aReg = 5.0/kFT[Nz-1]; cout << "highest value for kFT is " << kFT[Nz-1] << "/nm, and aReg = " << aReg << " nm" << endl;

  /* Positive particles are counted from j,k = 0 to N - 1; negative particles from j,k = N to 2N - 1. */
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (z[j]<0.0 and z[k]<0.0)  {uHT[i][j][k] = 2*pi* qpv[j]*qpv[j]*lB1*exp(-fabs(z[j]-z[k])*kFT[i])/kFT[i] + 2*pi* qpv[j]*qpv[j]*lB1*exp(-fabs(z[j]+z[k]-epsilon)*kFT[i])/kFT[i]*image; continue;}
    if (z[j]>=0.0 and z[k]>=0.0){uHT[i][j][k] = 2*pi* qpv[j]*qpv[j]*lB2*exp(-fabs(z[j]-z[k])*kFT[i])/kFT[i] + 2*pi* qpv[j]*qpv[j]*lB2*exp(-fabs(z[j]+z[k]-epsilon)*kFT[i])/kFT[i]*image; continue;}
    if (z[j]<0.0 and z[k]>=0.0) {uHT[i][j][k] = 2*pi* qpv[j]*qpv[k]*screen*exp(-fabs(z[j]-z[k])*kFT[i])/kFT[i]; continue;}
    if (z[j]>=0.0 and z[k]<0.0) {uHT[i][j][k] = 2*pi* qpv[j]*qpv[k]*screen*exp(-fabs(z[j]-z[k])*kFT[i])/kFT[i];}
  }}}
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (z[j]<0.0 and z[k]<0.0)  {uHT[i][j+Nz][k] = -2*pi* qpv[j]*qmv[j]*lB1*exp(-fabs(z[j]-z[k])*kFT[i])/kFT[i] - 2*pi* qpv[j]*qmv[j]*lB1*exp(-fabs(z[j]+z[k]-epsilon)*kFT[i])/kFT[i]*image; continue;}
    if (z[j]>=0.0 and z[k]>=0.0){uHT[i][j+Nz][k] = -2*pi* qpv[j]*qmv[j]*lB2*exp(-fabs(z[j]-z[k])*kFT[i])/kFT[i] - 2*pi* qpv[j]*qmv[j]*lB2*exp(-fabs(z[j]+z[k]-epsilon)*kFT[i])/kFT[i]*image; continue;}
    if (z[j]<0.0 and z[k]>=0.0) {uHT[i][j+Nz][k] = -2*pi* qpv[j]*qmv[k]*screen*exp(-fabs(z[j]-z[k])*kFT[i])/kFT[i]; continue;}
    if (z[j]>=0.0 and z[k]<0.0) {uHT[i][j+Nz][k] = -2*pi* qpv[j]*qmv[k]*screen*exp(-fabs(z[j]-z[k])*kFT[i])/kFT[i];}
  }}}
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
```

```
  uHT[i][j][k+Nz] = uHT[i][j+Nz][k];
}}}
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
  if (z[j]<0.0 and z[k]<0.0)  {uHT[i][j+Nz][k+Nz] = 2*pi* qmv[j]*qmv[j]*lB1*exp(-fabs(z[j]-z[k])*kFT[i])/kFT[i] + 2*pi* qmv[j]*qmv[j]*lB1*exp(-fabs(z[j]+z[k]-epsilon)*kFT[i])/kFT[i]*image; continue;}
  if (z[j]>=0.0 and z[k]>=0.0){uHT[i][j+Nz][k+Nz] = 2*pi* qmv[j]*qmv[j]*lB2*exp(-fabs(z[j]-z[k])*kFT[i])/kFT[i] + 2*pi* qmv[j]*qmv[j]*lB2*exp(-fabs(z[j]+z[k]-epsilon)*kFT[i])/kFT[i]*image; continue;}
  if (z[j]<0.0 and z[k]>=0.0) {uHT[i][j+Nz][k+Nz] = 2*pi* qmv[j]*qmv[k]*screen*exp(-fabs(z[j]-z[k])*kFT[i])/kFT[i]; continue;}
  if (z[j]>=0.0 and z[k]<0.0) {uHT[i][j+Nz][k+Nz] = 2*pi* qmv[j]*qmv[k]*screen*exp(-fabs(z[j]-z[k])*kFT[i])/kFT[i];}
}}}

/* And now the Regularized uHT: uRegHT */
for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
  uRegHT[i][j][k] = uHT[i][j][k];
  if (fabs(z[j]-z[k]) < aReg and (fabs(z[j]-0.5*epsilon) + fabs(z[k]-0.5*epsilon)) >= aReg) {
    if (z[j]<0.0 and z[k]<0.0)    {uRegHT[i][j][k] = 2*pi* qpv[j]*qpv[j]*lB1*exp(-aReg*kFT[i])/kFT[i] + 2*pi* qpv[j]*qpv[j]*lB1*exp(-fabs(z[j]+z[k]-epsilon)*kFT[i])/kFT[i]*image;}
    if (z[j]>=0.0 and z[k]>=0.0)  {uRegHT[i][j][k] = 2*pi* qpv[j]*qpv[j]*lB2*exp(-aReg*kFT[i])/kFT[i] + 2*pi* qpv[j]*qpv[j]*lB2*exp(-fabs(z[j]+z[k]-epsilon)*kFT[i])/kFT[i]*image;}
  }
  if (fabs(z[j]-z[k]) < aReg and (fabs(z[j]-0.5*epsilon) + fabs(z[k]-0.5*epsilon)) < aReg) {
    if (z[j]<0.0 and z[k]<0.0)    {uRegHT[i][j][k] = 2*pi* qpv[j]*qpv[j]*lB1*exp(-aReg*kFT[i])/kFT[i] + 2*pi* qpv[j]*qpv[j]*lB1*exp(-aReg*kFT[i])/kFT[i]*image;}
    if (z[j]>=0.0 and z[k]>=0.0)  {uRegHT[i][j][k] = 2*pi* qpv[j]*qpv[j]*lB2*exp(-aReg*kFT[i])/kFT[i] + 2*pi* qpv[j]*qpv[j]*lB2*exp(-aReg*kFT[i])/kFT[i]*image;}
    if (z[j]<0.0 and z[k]>=0.0)   {uRegHT[i][j][k] = 2*pi* qpv[j]*qpv[k]*screen*exp(-aReg*kFT[i])/kFT[i];}
    if (z[j]>=0.0 and z[k]<0.0)   {uRegHT[i][j][k] = 2*pi* qpv[j]*qpv[k]*screen*exp(-aReg*kFT[i])/kFT[i];}
  }
}}}

for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
  uRegHT[i][j+Nz][k] = uHT[i][j+Nz][k];
  if (fabs(z[j]-z[k]) < aReg and (fabs(z[j]-0.5*epsilon) + fabs(z[k]-0.5*epsilon)) >= aReg) {
    if (z[j]<0.0 and z[k]<0.0)    {uRegHT[i][j+Nz][k] = -2*pi* qpv[j]*qmv[j]*lB1*exp(-aReg*kFT[i])/kFT[i] - 2*pi* qpv[j]*qmv[j]*lB1*exp(-fabs(z[j]+z[k]-epsilon)*kFT[i])/kFT[i]*image;}
    if (z[j]>=0.0 and z[k]>=0.0)  {uRegHT[i][j+Nz][k] = -2*pi* qpv[j]*qmv[j]*lB2*exp(-aReg*kFT[i])/kFT[i] - 2*pi* qpv[j]*qmv[j]*lB2*exp(-fabs(z[j]+z[k]-epsilon)*kFT[i])/kFT[i]*image;}
  }
  if (fabs(z[j]-z[k]) < aReg and (fabs(z[j]-0.5*epsilon) + fabs(z[k]-0.5*epsilon)) < aReg) {
    if (z[j]<0.0 and z[k]<0.0)    {uRegHT[i][j+Nz][k] = -2*pi* qpv[j]*qmv[j]*lB1*exp(-aReg*kFT[i])/kFT[i] - 2*pi* qpv[j]*qmv[j]*lB1*exp(-aReg*kFT[i])/kFT[i]*image;}
    if (z[j]>=0.0 and z[k]>=0.0)  {uRegHT[i][j+Nz][k] = -2*pi* qpv[j]*qmv[j]*lB2*exp(-aReg*kFT[i])/kFT[i] - 2*pi* qpv[j]*qmv[j]*lB2*exp(-aReg*kFT[i])/kFT[i]*image;}
    if (z[j]<0.0 and z[k]>=0.0)   {uRegHT[i][j+Nz][k] = -2*pi* qpv[j]*qmv[k]*screen*exp(-aReg*kFT[i])/kFT[i];}
    if (z[j]>=0.0 and z[k]<0.0)   {uRegHT[i][j+Nz][k] = -2*pi* qpv[j]*qmv[k]*screen*exp(-aReg*kFT[i])/kFT[i];}
  }
  uRegHT[i][j][k+Nz] = uRegHT[i][j+Nz][k];
}}}

for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
  uRegHT[i][j+Nz][k+Nz] = uHT[i][j+Nz][k+Nz];
  if (fabs(z[j]-z[k]) < aReg and (fabs(z[j]-0.5*epsilon) + fabs(z[k]-0.5*epsilon)) >= aReg) {
    if (z[j]<0.0 and z[k]<0.0)    {uRegHT[i][j+Nz][k+Nz] = 2*pi* qmv[j]*qmv[j]*lB1*exp(-aReg*kFT[i])/kFT[i] + 2*pi* qmv[j]*qmv[j]*lB1*exp(-fabs(z[j]+z[k]-epsilon)*kFT[i])/kFT[i]*image;}
    if (z[j]>=0.0 and z[k]>=0.0)  {uRegHT[i][j+Nz][k+Nz] = 2*pi* qmv[j]*qmv[j]*lB2*exp(-aReg*kFT[i])/kFT[i] + 2*pi* qmv[j]*qmv[j]*lB2*exp(-fabs(z[j]+z[k]-epsilon)*kFT[i])/kFT[i]*image;}
  }
  if (fabs(z[j]-z[k]) < aReg and (fabs(z[j]-0.5*epsilon) + fabs(z[k]-0.5*epsilon)) < aReg) {
    if (z[j]<0.0 and z[k]<0.0)    {uRegHT[i][j+Nz][k+Nz] = 2*pi* qmv[j]*qmv[j]*lB1*exp(-aReg*kFT[i])/kFT[i] + 2*pi* qmv[j]*qmv[j]*lB1*exp(-aReg*kFT[i])/kFT[i]*image;}
    if (z[j]>=0.0 and z[k]>=0.0)  {uRegHT[i][j+Nz][k+Nz] = 2*pi* qmv[j]*qmv[j]*lB2*exp(-aReg*kFT[i])/kFT[i] + 2*pi* qmv[j]*qmv[j]*lB2*exp(-aReg*kFT[i])/kFT[i]*image;}
    if (z[j]<0.0 and z[k]>=0.0)   {uRegHT[i][j+Nz][k+Nz] = 2*pi* qmv[j]*qmv[k]*screen*exp(-aReg*kFT[i])/kFT[i];}
    if (z[j]>=0.0 and z[k]<0.0)   {uRegHT[i][j+Nz][k+Nz] = 2*pi* qmv[j]*qmv[k]*screen*exp(-aReg*kFT[i])/kFT[i];}
  }
}}}
```

```
}

void CoulombFluid::EvaluateReservoirMuEx (int medium) {

  cout << "MSA is used to solve OZ for the homogeneous system" << endl;
  //cout << "HNC is used to solve OZ for the homogeneous system" << endl;

  double I, c1, Ap, Am, Apm, lB, dr;
  double *dcpp,*tcpp,*hp,*dcpm,*tcpm,*hpm,*dcmm,*tcmm,*hm,*cnewp,*cnewm,*cnewpm,*Reg,*s,*s1,apm,var1,var2,var3,volp,volm,volpm;
  double error,tolerance=pow(10.0,-5.0),alpha2 = 0.03,turn=1.0, alphaMem;
  int iter=0,itermax = 30000, N;

  N = Nb; alphaMem = alpha2;

  dcpp = (double*) calloc (N,sizeof(double));  tcpp = (double*) calloc (N,sizeof(double)); hp = (double*) calloc (N,sizeof(double));
  dcpm = (double*) calloc (N,sizeof(double));  tcpm = (double*) calloc (N,sizeof(double)); hm = (double*) calloc (N,sizeof(double));
  dcmm = (double*) calloc (N,sizeof(double));  tcmm = (double*) calloc (N,sizeof(double)); hpm = (double*) calloc (N,sizeof(double));
  cnewp = (double*) calloc (N,sizeof(double)); cnewm = (double*) calloc (N,sizeof(double)); cnewpm = (double*) calloc (N,sizeof(double));
  Reg = (double*) calloc (N,sizeof(double));
  s = (double*) calloc (N,sizeof(double));  s1 = (double*) calloc (N,sizeof(double));

  cout << "Evaluating Reservoir MuEx for medium " << medium << endl;

  if (medium == 1) {c1 = cs1; I = max(10.0,8.0/kappa1); lB = lB1;}   if (medium == 2) {c1 = cs2; I = max(10.0,8.0/kappa2); lB = lB2;}
  if (medium != 1 and medium != 2) { cout << "Code can handle only 2 media. Value of medium = " << medium << " is not defined in algorithm." << endl; exit(1);  }

  for (i=0;i<N;i++) { s[i] = i*1.0/N; s1[i] = s[i]*I; }
  volp = pow(I,3.0)*c1*2*turn;  volm = pow(I,3.0)*c1*qp/qm*2*turn;  volpm = pow(I,3.0)*c1*sqrt(qp/qm)*2*turn;
  apm = (ap + am);
  Ap = qp*qp*lB/I;  Am = qm*qm*lB/I;  Apm = qp*qm*lB/I;

  for (i=0;i<N;i++) {    Reg[i] = exp(-3*s[i]*s[i]);  }

  error = 1.0;
  while ((error > tolerance and iter < itermax) or (turn < 1.0)) {
    error = 0.0;

    NR::sinft(tcpp,N);  for (i=1;i<N;i++) tcpp[i] *= (2.0/N);
    NR::sinft(tcmm,N);  for (i=1;i<N;i++) tcmm[i] *= (2.0/N);
    NR::sinft(tcpm,N);  for (i=1;i<N;i++) tcpm[i] *= (2.0/N);

    for (i=1;i<N;i++) cnewp[i] = tcpp[i]*tcmm[i]/i + tcpp[i] - tcpm[i]*tcpm[i]/i;
    for (i=1;i<N;i++) cnewm[i] = tcpp[i]*tcmm[i]/i + tcmm[i] - tcpm[i]*tcpm[i]/i;
    for (i=1;i<N;i++) cnewpm[i] = tcpm[i];
    for (i=1;i<N;i++) {
      var1 = (1.0+tcpp[i]/i)*(1.0 + tcmm[i]/i) - pow(tcpm[i]/i,2.0);
      cnewp[i] /= var1;
      cnewm[i] /= var1;
      cnewpm[i] /= var1;
    }
    NR::sinft(cnewp,N); NR::sinft(cnewm,N); NR::sinft(cnewpm,N);
```

```
    for (i=1;i<N;i++) {

      cnewp[i] = cnewp[i]/s[i]/volp;//+ Ap/s[i];
      if(s[i]*I/ap <= 2.0) dcpp[i] = alpha2*cnewp[i] + (1.0-alpha2)*dcpp[i];
      cnewm[i] = cnewm[i]/s[i]/volm;//+ Am/s[i];
      if(s[i]*I/am <= 2.0) dcmm[i] = alpha2*cnewm[i] + (1.0-alpha2)*dcmm[i];
      cnewpm[i] = cnewpm[i]/s[i]/volpm;//- Apm/s[i]
      if(s[i]*I/apm <= 1.0) dcpm[i] = alpha2*cnewpm[i] + (1.0-alpha2)*dcpm[i];

      if (s[i]*I/ap > 2.0)  cnewp[i]  = -Reg[i]*Ap/s[i]*turn;// + hp[i]  - log(hp[i]+1.0);  //HNC
      if (s[i]*I/am > 2.0)  cnewm[i]  = -Reg[i]*Am/s[i]*turn;// + hm[i]  - log(hm[i]+1.0);  //HNC
      if (s[i]*I/apm > 1.0) cnewpm[i] = Reg[i]*Apm/s[i]*turn;// + hpm[i] - log(hpm[i]+1.0); //HNC

      if (error < fabs(dcpp[i]-cnewp[i])) error = fabs(dcpp[i]-cnewp[i]);
      if (error < fabs(dcpm[i]-cnewpm[i])) error = fabs(dcpm[i]-cnewpm[i]);
    }
//    alpha2 = min(alphaMem/error,0.5); cout << "alpha2_c = " << alpha2 << endl;
    for (i=1;i<N;i++) {
      if(s[i]*I/ap > 2.0) { dcpp[i] = (1.0-alpha2)*dcpp[i] + alpha2*cnewp[i]; }
      cnewp[i] = dcpp[i]*s[i]*volp;

      if(s[i]*I/am > 2.0) { dcmm[i] = (1.0-alpha2)*dcmm[i] + alpha2*cnewm[i]; }
      cnewm[i] = dcmm[i]*s[i]*volm;

      if(s[i]*I/apm > 1.0) { dcpm[i] = (1.0-alpha2)*dcpm[i] + alpha2*cnewpm[i]; }
      cnewpm[i] = dcpm[i]*s[i]*volpm;
    }

    NR::sinft(cnewp,N);  for (i=1;i<N;i++) {cnewp[i] *= (2.0/N);};// cnewp[i] -= Ap/(i*pi)  *volp*2.0;}
    NR::sinft(cnewm,N);  for (i=1;i<N;i++) {cnewm[i] *= (2.0/N);};// cnewm[i] -= Am/(i*pi)  *volm*2.0;}
    NR::sinft(cnewpm,N); for (i=1;i<N;i++) {cnewpm[i]*= (2.0/N);};// cnewpm[i] += Apm/(i*pi)  *volpm*2.0;}

    for (i=1;i<N;i++) tcpp[i] = -cnewp[i]*cnewm[i]/i + cnewp[i] + cnewpm[i]*cnewpm[i]/i;
    for (i=1;i<N;i++) tcmm[i] = -cnewp[i]*cnewm[i]/i + cnewm[i] + cnewpm[i]*cnewpm[i]/i;
    for (i=1;i<N;i++) tcpm[i] = cnewpm[i];

    for (i=1;i<N;i++) {
      var2 =  (1.0-cnewp[i]/i)*(1.0 - cnewm[i]/i) - pow(cnewpm[i]/i,2.0);
      tcpp[i] /= var2;      tcmm[i] /= var2;      tcpm[i] /= var2;
    }
    NR::sinft(tcpp,N); NR::sinft(tcmm,N); NR::sinft(tcpm,N);

    for (i=1;i<N;i++) {
      cnewp[i] = tcpp[i]/s[i]/volp;      var1 = cnewp[i];
      cnewm[i] = tcmm[i]/s[i]/volm;      var2 = cnewm[i];
      cnewpm[i] = tcpm[i]/s[i]/volpm;    var3 = cnewpm[i];
      if (s[i]*I/ap <= 2.0) {cnewp[i] = -1.0;} if (s[i]*I/am <= 2.0) {cnewm[i] = -1.0;} if (s[i]*I/apm <= 1.0) {cnewpm[i] = -1.0;}
      if (error < fabs(var1-cnewp[i]))  error = fabs(var1-cnewp[i]) ;
      if (error < fabs(var2-cnewm[i]))  error = fabs(var2-cnewm[i]) ;
      if (error < fabs(var3-cnewpm[i])) error = fabs(var3-cnewpm[i]);
      hp[i]  = (1.0-alpha2)*var1 + alpha2*cnewp[i];     tcpp[i] = hp[i]*s[i]*volp;
      hm[i]  = (1.0-alpha2)*var2 + alpha2*cnewm[i];     tcmm[i] = hm[i]*s[i]*volm;
      hpm[i] = (1.0-alpha2)*var3 + alpha2*cnewpm[i];    tcpm[i] = hpm[i]*s[i]*volpm;
    }
    hp[0]=hm[0]=hpm[0]=-1.0;

    if (iter%1000 == 0) cout << "error = " << error << " at iteration " << iter << endl;
    if (iter%1000 == 0) {      RecordBulkCorrelationFunctions(s1,hp,hm,hpm,N,medium);    }

    if (turn < 1.0) {volp/=turn;volm/=turn;volpm/=turn; turn += 5e-4; volp*=turn; volm*=turn; volpm*=turn;}

    iter++;
  }

  cout << "error = " << error << " at iteration " << iter << endl;

  RecordBulkCorrelationFunctions(s1,hp,hm,hpm,N,medium);

  if (i == itermax) cout << "Maximum number of iterations reached." << endl;

  if (medium == 1) { muEx1p = 0.0; muEx1m = 0.0;
    for (i=1;i<N/2;i++) {
      dr = 4.0*pi*(pow(s[i]*I,3.0)-pow(s[i-1]*I,3.0))/3.0;
//      dr = 4.0*pi*pow(s[i]*I,2.0)*(s[i]-s[i-1])*I;
      if (s[i]*I/ap > 0.0) muEx1p += c1*(0.5*hp[i]*(hp[i]-dcpp[i])-dcpp[i])* dr;
      if (s[i]*I/apm > 0.0) muEx1p += c1*qp/qm*(0.5*hpm[i]*(hpm[i]-dcpm[i])-dcpm[i])* dr;
      if (s[i]*I/am > 0.0) muEx1m += c1*qp/qm*(0.5*hm[i]*(hm[i]-dcmm[i])-dcmm[i])* dr;
      if (s[i]*I/apm > 0.0) muEx1m += c1*(0.5*hpm[i]*(hpm[i]-dcpm[i])-dcpm[i])* dr;
    }
    cout << "muEx+ = " << muEx1p << " and muEx- = " << muEx1m << " in medium " << medium << endl;
  }
  if (medium == 2) { muEx2p = 0.0; muEx2m = 0.0;
    for (i=1;i<N/2;i++) {
      dr = 0.75*pi*(pow(s[i]*I,3.0)-pow(s[i-1]*I,3.0));
      if (s[i]*I/ap > 0.0) muEx2p += 0.5*c1*(hp[i]*(hp[i]-dcpp[i])-dcpp[i])* dr;
      if (s[i]*I/apm > 0.0) muEx2p += 0.5*c1*qp/qm*(hpm[i]*(hpm[i]-dcpm[i])-dcpm[i])* dr;
      if (s[i]*I/am > 0.0) muEx2m += 0.5*c1*qp/qm*(hm[i]*(hm[i]-dcmm[i])-dcmm[i])* dr;
      if (s[i]*I/apm > 0.0) muEx2m += 0.5*c1*(hpm[i]*(hpm[i]-dcpm[i])-dcpm[i])* dr;
    }
    cout << "muEx+ = " << muEx2p << " and muEx- = " << muEx2m << " in medium " << medium << endl;
  }

//  InitiateCorrelationFunctions(hp,hm,hpm,dcpp,dcmm,dcpm,s1,N,medium);
  InitiateCorrelationFunctions2Walls(hp,hm,hpm,dcpp,dcmm,dcpm,s1,N,medium);
  cout << "Correlation functions are initialized for a single electrolyte solution between 2 walls." << endl;
  ResetQs();

/* TEST muEx */  /*
  double * mu;
  mu = (double*) calloc (2*Nz,sizeof(double));
  for (j=0;j<2*Nz;j++) {
    for (i=1;i<Nr/2;i++) {
      mu[j] += pi*(r[i]*r[i]-r[i-1]*r[i-1])*(0.5*h[i][j][Nz/2]*(h[i][j][Nz/2] - c[i][j][Nz/2]) - c[i][j][Nz/2]);
    }
  }
```

```
  muEx1p = 0.0; muEx1m = 0.0;
  for (j=0;j<Nz;j++) {
    muEx1p += c1*mu[j]*dz[j];
    muEx1m += c1*qp/qm*mu[j]*dz[j];
  }
  for (j=0;j<Nz;j++) {
    muEx1p += c1*qp/qm*mu[j+Nz]*dz[j];
    muEx1m += c1*mu[j+Nz]*dz[j];
  }
//  muEx1m = muEx1p;
  cout << "muEx+ = " << muEx1p << " and muEx- = " << muEx1m << " in medium " << medium << endl;
  free (mu);
*/

  gsl_matrix * m = gsl_matrix_alloc (2, 2);
  gsl_permutation * p = gsl_permutation_alloc (2);
  int sign[1];
  double vol, det, entropy, Ftot, dist;

  double muEx;

  hFT.assign (0.0,Nr,3*Nz,3*Nz);
  vol = pow(r[Nr-1],3.0);

  mu.assign(0.0,3*Nz,3*Nz);



  for (i=1;i<Nr;i++) {
    dist = r[i];
    if (dist < 2*ap) continue;
    if (dist > 4.0/kappa1) continue;
      muEx -= 4*pi*r[i]*r[i]*(r[i] - r[i-1]) * (0.5*hp[i]*hp[i] + hp[i] + 0.5*hpm[i]*hpm[i] + hpm[i] + 0.5*hm[i]*hm[i] + hm[i]);
      muEx += 4*pi*r[i]*r[i]*(r[i] - r[i-1]) * ( (hp[i]+1.0)*log(hp[i]+1.0+1e-33) + (hp[i]+1.0)*(Reg[i]*Ap/s[i]) );
      muEx += 4*pi*r[i]*r[i]*(r[i] - r[i-1]) * ( (hpm[i]+1.0)*log(hpm[i]+1.0+1e-33) + (hpm[i]+1.0)*(-Reg[i]*Apm/s[i]) );
      muEx += 4*pi*r[i]*r[i]*(r[i] - r[i-1]) * ( (hm[i]+1.0)*log(hm[i]+1.0+1e-33) + (hm[i]+1.0)*(Reg[i]*Am/s[i]) );
  }


  for (j=0;j<3*Nz;j++) {
    for (k=0;k<3*Nz;k++) {
      Fref += 0.5*c1*c1*muEx;
    }
    //    Fexc[nf] += 0.5*n[j]*(c[1][j][j]-Qs[1][j][j]-uReg[1][j][j]);
  }


  for (i=0;i<Nr;i++) { dc[i] = hp[i]*s[i]*vol; }    NR::sinft(dc,Nr);  for (i=1;i<Nr;i++) dc[i] *= (2.0/Nr);  for (i=1;i<Nr;i++) { hFT[i][0][0] = dc[i]/i;}
  for (i=0;i<Nr;i++) { dc[i] = hpm[i]*s[i]*vol; }   NR::sinft(dc,Nr);  for (i=1;i<Nr;i++) dc[i] *= (2.0/Nr);  for (i=1;i<Nr;i++) { hFT[i][1][0] = hFT[i][0][1] = dc[i]/i;}
  for (i=0;i<Nr;i++) { dc[i] = hm[i]*s[i]*vol; }    NR::sinft(dc,Nr);  for (i=1;i<Nr;i++) dc[i] *= (2.0/Nr);  for (i=1;i<Nr;i++) { hFT[i][1][1] = dc[i]/i;}
  for (i=1;i<Nr;i++) {
    for (j=0;j<2;j++){      for (k=0;k<2;k++){
      gsl_matrix_set (m, j, k, delta[j][k] + hFT[i][j][k]*n[k]);
    }}
    gsl_linalg_LU_decomp(m,p,sign);
    det = gsl_linalg_LU_lndet (m);
    for (j=0;j<2;j++) det -= c1*hFT[i][j][j];
  //    Fexc[nf] -= 0.5/pow(2*pi,3.0)*4*pi*i*i*det;
    Fref -= 0.5/vol*4*pi*i*i*det *0.25; /* 1/vol by normalization of k and factor 0.25 from discrete FT*/
  }


  entropy = c1;
  Ftot = (Fref+entropy)/c1;

  free(s); gsl_matrix_free (m); gsl_permutation_free (p);



  ofstream datafile;
  char filename[50];

  strcpy(filename,"Data/"); strcat(filename, project); strcat(filename,"_Info.dat");

  datafile.open(filename,ios::app);
  datafile << "\n\tReservoir excess chemical potential muEx+ = " << muEx1p << " kBT, and muEx- = " << muEx1m << " kBT in medium " << medium << endl;
  datafile.close();

  free (dcpp); free (cnewp); free (tcpp); free (hp);  free (dcpm); free (cnewm); free (tcpm); free (hpm); free(Reg); free (dcmm); free (cnewpm); free (tcmm); free (hm);
}

void CoulombFluid::OZ_HNC() {

  double I = r[Nr-1], alpha2 = alpha;
  gsl_dht * t = gsl_dht_new(Nr, 0.0, I);
  gsl_matrix * m = gsl_matrix_alloc (2*Nz, 2*Nz),* m2 = gsl_matrix_alloc (2*Nz, 2*Nz);
  gsl_vector * hvec = gsl_vector_alloc (2*Nz), * cvec = gsl_vector_alloc (2*Nz);
  gsl_permutation * p = gsl_permutation_alloc (2*Nz);
  int sign[1], it=0;
  char answer;

  alpha2 = 0.001;

//  *sign=1; /* just to initialize the variable; not necessary */

  errmax = 1e9;
  while (errmax > tolerance*1e2) {
  errmax = 0.0; it++;

  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (r[i]*r[i] + pow(z[j]-z[k],2.0) < 4*ap*ap) continue;
    newc[i][j][k] = - u[i][j][k]+h[i][j][k] - log(h[i][j][k] + 1.0); /* actually c + u: u can be transformed analytically, so is subtracted after the numerical HT */
```

```
  }}}
  for (i=0;i<Nr;i++){    for (j=0;j<Nz;j++){      for (k=Nz;k<2*Nz;k++){
    if (r[i]*r[i] + pow(z[j]-z[k-Nz],2.0) < pow(ap+am,2.0)) continue;
    newc[i][j][k] = - u[i][j][k]+h[i][j][k] - log(h[i][j][k] + 1.0);
    newc[i][j+Nz][k-Nz] = newc[i][j][k];
  }}}
  for (i=0;i<Nr;i++){    for (j=Nz;j<2*Nz;j++){      for (k=Nz;k<2*Nz;k++){
    if (r[i]*r[i] + pow(z[j-Nz]-z[k-Nz],2.0) < 4*am*am) continue;
    newc[i][j][k] = - u[i][j][k]+ h[i][j][k] - log(h[i][j][k] + 1.0);
  }}}

  for (i=0;i<Nr;i++){  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    err = c[i][j][k]-newc[i][j][k];
    if (err != err) {
      cout << "singularity detected at (r,z,z') = (" << r[i] << "," << z[j]  << "," << z[k] << ")" << endl;
      if (!Overrule) { cout << "Press y to continue" << endl; cin >> answer; if (answer == 'y') Overrule = true;}
      if (!Overrule) exit(1);
    }
    if (fabs(err) > errmax) errmax = fabs(err);
    c[i][j][k] -= alpha2*err; // alpha = 1.0 means complete replacement with new data; 0 means no replacement (does not converge)
    newc[i][j][k] = c[i][j][k];
  }}}
  /* Bessel transform the direct correlation function */
  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
      for (i=0;i<Nr;i++) { dc[i] = newc[i][j][k]; }
      gsl_dht_apply(t, dc, dcHT); for (i=0;i<Nr;i++) dcHT[i] *= gsl_sf_bessel_zero_J0(Nr+1)/I/I;
      for (i=0;i<Nr;i++) { newc[i][j][k] = dcHT[i];}//-uHT[i][j][k]; }
  }}
  /* Solving M H(k) = C(k), transforming H(k) into H(r), and correcting H for r < hard core */

  for (i=0;i<Nr;i++) {
    for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){ gsl_matrix_set (m, j, k, delta[j][k] + newc[i][j][k]*n[k]);}}// cout << gsl_matrix_get (m, j, k) << endl; }}
    gsl_linalg_LU_decomp(m,p,sign);
    for (k=0;k<2*Nz;k++){
      for (j=0;j<2*Nz;j++) gsl_vector_set (cvec,j,newc[i][j][k]);
      gsl_linalg_LU_solve(m,p,cvec,hvec);
      for (j=0;j<2*Nz;j++) newh[i][j][k] = gsl_vector_get (hvec,j);
    }
  }
  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    for (i=0;i<Nr;i++) tcHT[i] = newh[i][j][k];
    gsl_dht_apply(t, tcHT, tc); for (i=0;i<Nr;i++) tc[i] *= gsl_sf_bessel_zero_J0(Nr+1)/I/I;
    for (i=0;i<Nr;i++) newh[i][j][k] = tc[i];
  }}

  for (i=0;i<Nr;i++) {
    for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
//      if (r[i]*r[i] + pow(z[j]-z[k],2.0) > 4*ap*ap) h[i][j][k] = newh[i][j][k];
      if (r[i]*r[i] + pow(z[j]-z[k],2.0) < 4*ap*ap) newh[i][j][k] = - 1.0;
    }}
    for (j=0;j<Nz;j++){      for (k=Nz;k<2*Nz;k++){
//      if (r[i]*r[i] + pow(z[j]-z[k-Nz],2.0) > pow(ap+am,2.0)) { h[i][j][k] = newh[i][j][k]; h[i][j+Nz][k-Nz] = h[i][j][k]; }
      if (r[i]*r[i] + pow(z[j]-z[k],2.0) < pow(ap+am,2.0))  newh[i][j][k] = - 1.0;      newh[i][j+Nz][k-Nz] = newh[i][j][k];
    }}
    for (j=Nz;j<2*Nz;j++){   for (k=Nz;k<2*Nz;k++){
//      if (r[i]*r[i] + pow(z[j-Nz]-z[k-Nz],2.0) > 4*am*am) h[i][j][k] = newh[i][j][k];
      if (r[i]*r[i] + pow(z[j-Nz]-z[k-Nz],2.0) < 4*am*am) newh[i][j][k] = - 1.0;
    }}

    for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
      err = h[i][j][k]-newh[i][j][k];
      if (fabs(err) > errmax) errmax = fabs(err); /* wARNING: watch out what is happening in the next line */
      h[i][j][k] -= alpha2*err; // alpha = 1.0 means complete replacement with new data; 0 means no correction
    }}
  }
  /* Solving M newC(k) = H(k) [other M as above], and transforming newC(k) into newC(r) */

  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    for (i=0;i<Nr;i++) tc[i] = h[i][j][k];
    gsl_dht_apply(t, tc, tcHT); for (i=0;i<Nr;i++) tcHT[i] *= gsl_sf_bessel_zero_J0(Nr+1)/I/I;
    for (i=0;i<Nr;i++) newh[i][j][k] = tcHT[i];
  }}
  for (i=0;i<Nr;i++) {

    for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){ gsl_matrix_set (m, j, k, n[k]*delta[j][k] + n[j]*newh[i][j][k]*n[k]);    }}
    gsl_linalg_LU_decomp(m,p,sign);
    for (k=0;k<2*Nz;k++){
      for (j=0;j<2*Nz;j++) gsl_vector_set (cvec,j,delta[j][k]);
      gsl_linalg_LU_solve(m,p,cvec,hvec);
      for (j=0;j<2*Nz;j++) gsl_matrix_set(m2,j,k,gsl_vector_get (hvec,j));
    }
    for (k=0;k<2*Nz;k++){
      for (j=0;j<2*Nz;j++){
    newc[i][j][k] = delta[j][k]/n[k] - gsl_matrix_get(m2,j,k);
    }}
  }
  /* (inverse) HT of newc_ij(k) into newc_ij(r) */
  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    for (i=0;i<Nr;i++) dcHT[i] = newc[i][j][k];
    gsl_dht_apply(t, dcHT, dc); for (i=0;i<Nr;i++) dc[i] *= gsl_sf_bessel_zero_J0(Nr+1)/I/I;
    for (i=0;i<Nr;i++) newc[i][j][k] = dc[i];
  }}
  cout << "Iteration " << it << "  Maximum error " << errmax << endl;
  if (it%1 == 0) RecordCorrelationFunctions();
  }
  gsl_matrix_free (m); gsl_vector_free (hvec); gsl_vector_free (cvec); gsl_dht_free (t); gsl_permutation_free (p);
}

void CoulombFluid::InitialCorrelationFunctions() {

 for (j=0;j<2*Nz;j++) { for (k=0;k<2*Nz;k++) {  for (i=0;i<Nr;i++) {
    hInit[i][j][k] = h[i][j][k];
    cInit[i][j][k] = c[i][j][k];
 }}}
 for (j=0;j<Nz;j++) {
```

```
   cpInit[j] = cp[j];
   cmInit[j] = cm[j];
   n[j] = cp[j]*dz[j];
 }
 for (j=Nz;j<2*Nz;j++) { n[j] = cm[j-Nz]*dz[j-Nz]; }

}

void CoulombFluid::SetCorrelationFunctions() {

 for (j=0;j<2*Nz;j++) { for (k=0;k<2*Nz;k++) {  for (i=0;i<Nr;i++) {
    h[i][j][k] = hInit[i][j][k];
    c[i][j][k] = cInit[i][j][k];
 }}}
 for (j=0;j<Nz;j++) {
   cp[j] = cpInit[j];
   cm[j] = cmInit[j];
   n[j] = cp[j]*dz[j];
 }
 for (j=Nz;j<2*Nz;j++) { n[j] = cm[j-Nz]*dz[j-Nz]; }

}

void CoulombFluid::FinalCorrelationFunctions() {

 for (j=0;j<2*Nz;j++) { for (k=0;k<2*Nz;k++) {  for (i=0;i<Nr;i++) {
    hFin[i][j][k] = h[i][j][k];
    cFin[i][j][k] = c[i][j][k];
 }}}
 for (j=0;j<Nz;j++) {
   cpFin[j] = cp[j];
   cmFin[j] = cm[j];
   n[j] = cp[j]*dz[j];
 }
 for (j=Nz;j<2*Nz;j++) { n[j] = cm[j-Nz]*dz[j-Nz]; }

 alpha0[1] = alpha;
 errmax0=errmax;

}

void CoulombFluid::GetCorrelationFunctions() {

 for (j=0;j<2*Nz;j++) { for (k=0;k<2*Nz;k++) {  for (i=0;i<Nr;i++) {
    h[i][j][k] = hFin[i][j][k];
    c[i][j][k] = cFin[i][j][k];
 }}}
 for (j=0;j<Nz;j++) {
   cp[j] = cpFin[j];
   cm[j] = cmFin[j];
   n[j] = cp[j]*dz[j];
 }
 for (j=Nz;j<2*Nz;j++) { n[j] = cm[j-Nz]*dz[j-Nz]; }
 alpha0[0]=alpha0[1];
 errmax=errmax0;

}

void CoulombFluid::ResetQs() {
// Determine Qs, and the DHT of Qs (QsHT), and adapt h

  double d, deltah, deltahp;
  int l;


  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (fabs(z[j]-z[k]) >= apv[j]+apv[k]) continue;
    d = sqrt(pow(apv[j]+apv[k],2.0) - pow(z[j]-z[k],2.0));
    i=0;    while (r[i] < d and i<Nr-3) i++; l = i;

    deltah = h[l][j][k] - (h[l+1][j][k]-h[l][j][k])/(r[l+1]-r[l])*(r[l]-d) + 1.0;
    deltahp = (h[l+1][j][k]-h[l][j][k])/(r[l+1]-r[l]);// - ( (h[l+2][j][k]-h[l+1][j][k])/(r[l+2]-r[l+1]) - (h[l+1][j][k]-h[l][j][k])/(r[l+1]-r[l]) ) / (r[l+2]-r[l+1]) * (r[l]-d);

    for (i=0;i<l;i++) Qs[i][j][k] = deltah + 0.5*(r[i]*r[i] - d*d)*deltahp/d;
    for (i=0;i<Nr;i++){
      if (r[i] == 0.0) {QsHT[i][j][k] = Large; continue;}
      QsHT[i][j][k] = d*(deltah*gsl_sf_bessel_J1(d*kFT[i])/kFT[i] - deltahp*gsl_sf_bessel_Jn(2,d*kFT[i])/kFT[i]/kFT[i]);
      QsHT[i][j][k] *= 2*pi;
    }
    for (i=0;i<l;i++) h[i][j][k] = Qs[i][j][k]-1.0;
  }}
  for (j=0;j<Nz;j++){      for (k=Nz;k<2*Nz;k++){
    if (fabs(z[j]-z[k-Nz]) >= apv[j]+amv[k-Nz]) continue;
    d = sqrt(pow(apv[j]+amv[k-Nz],2.0) - pow(z[j]-z[k-Nz],2.0));
    i=0;    while (r[i] < d and i<Nr-3) i++; l = i;

    deltah = h[l][j][k] - (h[l+1][j][k]-h[l][j][k])/(r[l+1]-r[l])*(r[l]-d) + 1.0;
    deltahp = (h[l+1][j][k]-h[l][j][k])/(r[l+1]-r[l]);// - ( (h[l+2][j][k]-h[l+1][j][k])/(r[l+2]-r[l+1]) - (h[l+1][j][k]-h[l][j][k])/(r[l+1]-r[l]) ) / (r[l+2]-r[l+1]) * (r[l]-d);

    for (i=0;i<l;i++) { Qs[i][j][k] = deltah + 0.5*(r[i]*r[i] - d*d)*deltahp/d; }
    for (i=0;i<Nr;i++){
      if (r[i] == 0.0) {QsHT[i][j][k] = Large; continue;}
      QsHT[i][j][k] = d*(deltah*gsl_sf_bessel_J1(d*kFT[i])/kFT[i] - deltahp*gsl_sf_bessel_Jn(2,d*kFT[i])/kFT[i]/kFT[i]);
      QsHT[i][j][k] *= 2*pi;
    }
    for (i=0;i<l;i++) h[i][j][k] = Qs[i][j][k]-1.0;
  }}
  for (j=Nz;j<2*Nz;j++){      for (k=0;k<Nz;k++){
    if (fabs(z[j-Nz]-z[k]) >= apv[j-Nz]+amv[k]) continue;
    d = sqrt(pow(apv[j-Nz]+amv[k],2.0) - pow(z[j-Nz]-z[k],2.0));
    i=0;    while (r[i] < d and i<Nr-3) i++; l = i;

    deltah = h[l][j][k] - (h[l+1][j][k]-h[l][j][k])/(r[l+1]-r[l])*(r[l]-d) + 1.0;
    deltahp = (h[l+1][j][k]-h[l][j][k])/(r[l+1]-r[l]);// - ( (h[l+2][j][k]-h[l+1][j][k])/(r[l+2]-r[l+1]) - (h[l+1][j][k]-h[l][j][k])/(r[l+1]-r[l]) ) / (r[l+2]-r[l+1]) * (r[l]-d);
```

```
    for (i=0;i<l;i++) { Qs[i][j][k] = deltah + 0.5*(r[i]*r[i] - d*d)*deltahp/d; }
    for (i=0;i<Nr;i++){
      if (r[i] == 0.0) {QsHT[i][j][k] = Large; continue;}
      QsHT[i][j][k] = d*(deltah*gsl_sf_bessel_J1(d*kFT[i])/kFT[i] - deltahp*gsl_sf_bessel_Jn(2,d*kFT[i])/kFT[i]/kFT[i]);
      QsHT[i][j][k] *= 2*pi;
    }
    for (i=0;i<l;i++) h[i][j][k] = Qs[i][j][k]-1.0;
  }}
  for (j=Nz;j<2*Nz;j++){      for (k=Nz;k<2*Nz;k++){
    if (fabs(z[j-Nz]-z[k-Nz]) >= amv[j-Nz]+amv[k-Nz]) continue;
    d = sqrt(pow(amv[j-Nz]+amv[k-Nz],2.0) - pow(z[j-Nz]-z[k-Nz],2.0));
    i=0;    while (r[i] < d and i<Nr-3) i++; l = i;

    deltah = h[l][j][k] - (h[l+1][j][k]-h[l][j][k])/(r[l+1]-r[l])*(r[l]-d) + 1.0;
    deltahp = (h[l+1][j][k]-h[l][j][k])/(r[l+1]-r[l]);// - ( (h[l+2][j][k]-h[l+1][j][k])/(r[l+2]-r[l+1]) - (h[l+1][j][k]-h[l][j][k])/(r[l+1]-r[l]) ) / (r[l+2]-r[l+1]) * (r[l]-d);

    for (i=0;i<l;i++) Qs[i][j][k] = deltah + 0.5*(r[i]*r[i] - d*d)*deltahp/d;
    for (i=0;i<Nr;i++){
      if (r[i] == 0.0) {QsHT[i][j][k] = Large; continue;}
      QsHT[i][j][k] = d*(deltah*gsl_sf_bessel_J1(d*kFT[i])/kFT[i] - deltahp*gsl_sf_bessel_Jn(2,d*kFT[i])/kFT[i]/kFT[i]);
      QsHT[i][j][k] *= 2*pi;
    }
    for (i=0;i<l;i++) h[i][j][k] = Qs[i][j][k]-1.0;
  }}


  // ALERT Qs and QsHT are set to zero
  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    d = sqrt(pow(amv[j%Nz]+amv[k%Nz],2.0) - pow(z[j%Nz]-z[k%Nz],2.0));
    i=0;    while (r[i] < d and i<Nr-3) i++; l = i;

    for (i=0;i<Nr;i++) Qs[i][j][k] = 0.0;
    for (i=0;i<Nr;i++) QsHT[i][j][k] = 0.0;
    for (i=0;i<l;i++) h[i][j][k] = Qs[i][j][k]-1.0;
  }}


}

void CoulombFluid::OZ_HNC_ShortRange() {

  double I = r[Nr-1], alpha2 = alpha, alpha3 = alpha, b, d;
  gsl_dht * t = gsl_dht_new(Nr, 0.0, I);
  gsl_matrix * m = gsl_matrix_alloc (2*Nz, 2*Nz),* m2 = gsl_matrix_alloc (2*Nz, 2*Nz);
  gsl_vector * hvec = gsl_vector_alloc (2*Nz), * cvec = gsl_vector_alloc (2*Nz);
  gsl_permutation * p = gsl_permutation_alloc (2*Nz);
  int sign[1], it=0;

  alpha2 = 0.5;  alpha3 = 1.0;
  b = gsl_sf_bessel_zero_J0(Nr+1)/I/I;

  ResetQs();

  RecordCorrelationFunctions();

  errmax = 1e9;
  while (errmax > 1e-1) {
  errmax = 0.0; it++;


  ResetQs();

  // Determine C*(k) from (H*-Y*)(r)
  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
      for (i=0;i<Nr;i++) { dc[i] = h[i][j][k]-y[i][j][k]; }
      gsl_dht_apply(t, dc, dcHT);
      for (i=0;i<Nr;i++) { c[i][j][k] = dcHT[i];}
  }}

// Determine new Y*(k)
// inversion of ( P - P (C-U-Q)P) for every i, stored in newh[i][j][k] (dummy tensor, just like cvec, hvec, dc, and dcHT here)

  for (i=0;i<Nr;i++) {
    for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
      gsl_matrix_set (m, j, k, delta[j][k]*n[k] - n[j]*(c[i][j][k]-uRegHT[i][j][k]-QsHT[i][j][k])*n[k]);
    }}
    gsl_linalg_LU_decomp(m,p,sign);
    for (k=0;k<2*Nz;k++){
      for (j=0;j<2*Nz;j++) gsl_vector_set (cvec,j,delta[j][k]);
      gsl_linalg_LU_solve(m,p,cvec,hvec);
      for (j=0;j<2*Nz;j++) newh[i][j][k] = gsl_vector_get (hvec,j);
    }
  }

  for (i=0;i<Nr;i++) {  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    newh[i][j][k] += - delta[j][k]/n[j] - c[i][j][k] + QsHT[i][j][k];
  }}}

// Y*(k) -> Y*(r)

  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
      for (i=0;i<Nr;i++) { dc[i] = newh[i][j][k]; }
      gsl_dht_apply(t, dc, dcHT); for (i=0;i<Nr;i++) dcHT[i] *= b*b*(2*pi);
      for (i=0;i<Nr;i++) { newh[i][j][k] = dcHT[i];}
  }}

  for (i=0;i<Nr;i++) {  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    y[i][j][k] = (1.0-alpha3)*y[i][j][k] + alpha3*newh[i][j][k];
  }}}

// new H*(r) from Y*(r)

  for (i=0;i<Nr;i++) {  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    newh[i][j][k] = exp(y[i][j][k]-u[i][j][k]+uReg[i][j][k]) -1.0;
```

```
//    newh[i][j][k] = -u[i][j][k]+uReg[i][j][k]; //ALERt
//    newh[i][j][k] = exp(-kappa1*sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0)))/sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
  }}}
  for (i=0;i<Nr;i++) {  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    d = sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
    if (d >=  2*ap) {err = h[i][j][k]-newh[i][j][k]; if (fabs(err) > errmax) errmax = fabs(err); h[i][j][k] -= alpha2*err;}
    if (d >=  2*am) {err = h[i][j+Nz][k+Nz]-newh[i][j+Nz][k+Nz]; if (fabs(err) > errmax) errmax = fabs(err); h[i][j+Nz][k+Nz] -= alpha2*err;}
    if (d >= ap+am) {err = h[i][j][k+Nz]-newh[i][j][k+Nz]; if (fabs(err) > errmax) errmax = fabs(err); h[i][j][k+Nz] -= alpha2*err; h[i][j+Nz][k] = h[i][j][k+Nz];}
  }}}

  cout << "Iteration " << it << "  Maximum error " << errmax << endl;
  if (it%1 == 0 and errmax != 0) RecordCorrelationFunctions();
  if (it>50) errmax = 0;

  } /* end of loop here */
  for (i=0;i<Nr;i++) {  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    c[i][j][k] = h[i][j][k]-y[i][j][k];
  }}}

  cout << setw(20) << it << " iterations after " << evaluations << " evaluations" << endl;
  if (it == 1 and evaluations > 13) { convergence = true; cout << "Convergence to requested precision." << endl; }
  evaluations++;

  if (errmax != 0) RecordCorrelationFunctions();

  gsl_matrix_free (m); gsl_matrix_free (m2); gsl_vector_free (hvec); gsl_vector_free (cvec); gsl_dht_free (t); gsl_permutation_free (p);
}

void CoulombFluid::OZ_HNC_Alternative() {

  double I = r[Nr-1], alpha2 = alpha, alpha3 = alpha, alphaMem, b, d;
  gsl_dht * t = gsl_dht_new(Nr, 0.0, I);
  gsl_matrix * m = gsl_matrix_alloc (2*Nz, 2*Nz),* m2 = gsl_matrix_alloc (2*Nz, 2*Nz);
  gsl_vector * hvec = gsl_vector_alloc (2*Nz), * cvec = gsl_vector_alloc (2*Nz);
  gsl_permutation * p = gsl_permutation_alloc (2*Nz);
  int sign[1], it=0, imax, jmax, kmax, na;
  char pair[3];
  bool peak = false;

  alpha2 = 0.5;  alpha3 = 1.0; alphaMem = alpha2;
  b = gsl_sf_bessel_zero_J0(Nr+1)/I/I;

  ResetQs();

  RecordCorrelationFunctions();

  errmax = 1e9;
  while (errmax > 1e-3) {

    for (na=0;na<1;na++) {

      errmax = -1e9; it++;

  /* Iteration according to Ng, J. Chem. Phys. 61, 2680 (1974), instead of Picard. Not too successful, so replaced. Present in earlier versions <= 010 */

  // new H*(r) from C*(r)
  for (i=0;i<Nr;i++) {  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
newh[i][j][k] = exp(h[i][j][k] - c[i][j][k]-u[i][j][k]+uReg[i][j][k]) -1.0;
if (newh[i][j][k] > 1e2 and sqrt(r[i]*r[i]+pow(z[j]-z[k],2.0)) > ap+am) { peak = true; newh[i][j][k] = c[i][j][k]+u[i][j][k]-uReg[i][j][k] + 1.0;}
  }}}
  if (peak) { cout << " ALERT attenuation of new total correlation function " << endl; peak = false; }

  if(BulkPhase2) {
  //    for (i=0;i<Nr;i++) {  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
  //      if ( (z[Nz-1]-z[j%Nz]) < BulkCutoff or (z[Nz-1]-z[k%Nz]) < BulkCutoff ) newh[i][j][k] = newh[i][max(j-1,0)][max(k-1,0)];
  //      if ( (z[Nz-1]-z[j%Nz]) < BulkCutoff or (z[Nz-1]-z[k%Nz]) < BulkCutoff ) newh[i][j][k] = h[i][j][k];
  //    }}}
  }

  for (i=0;i<Nr;i++) {  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
d = sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
if (d >=  apv[j]+apv[k]) {err = h[i][j][k]-newh[i][j][k]; if (fabs(err) > errmax) {errmax = fabs(err);imax=i;jmax=j;kmax=k;strcpy(pair,"pp");} }
 if (d >=  amv[j]+amv[k]) {err = h[i][j+Nz][k+Nz]-newh[i][j+Nz][k+Nz]; if (fabs(err) > errmax) {errmax = fabs(err);imax=i;jmax=j;kmax=k;strcpy(pair,"mm");} }
if (d >=  apv[j]+amv[k]) {err = h[i][j][k+Nz]-newh[i][j][k+Nz]; if (fabs(err) > errmax) {errmax = fabs(err);imax=i;jmax=j;kmax=k;strcpy(pair,"pm");} }
  }}}

  //  cout << "alpha2 = " << alpha2 << " and new alpha = " << min(alphaMem / errmax,0.5) << endl;
  //  if (alphaLast <= min(alphaMem / errmax,0.5) or 0.5*alphaLast > min(alphaMem / errmax,0.5))
  alpha2 = min(alphaMem / errmax,0.5);
  //  else {cout << "division" << endl; alpha2 *= 0.25;}
  cout << "alpha2 = " << alpha2 << endl;
  //  alphaLast = min(alphaMem / errmax,0.5);

  for (i=0;i<Nr;i++) {  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
d = sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
if (d >=  apv[j]+apv[k]) {err = h[i][j][k]-newh[i][j][k]; h[i][j][k] -= alpha2*err;}
if (d >=  amv[j]+amv[k]) {err = h[i][j+Nz][k+Nz]-newh[i][j+Nz][k+Nz]; h[i][j+Nz][k+Nz] -= alpha2*err;}
if (d >=  apv[j]+amv[k]) {err = h[i][j][k+Nz]-newh[i][j][k+Nz]; h[i][j][k+Nz] -= alpha2*err; h[i][j+Nz][k] = h[i][j][k+Nz];}
  }}}


  ResetQs();

  // Determine C*(k) from (H*-Y*)(r)
  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
for (i=0;i<Nr;i++) { dc[i] = h[i][j][k]; }
gsl_dht_apply(t, dc, dcHT);
for (i=0;i<Nr;i++) { newh[i][j][k] = dcHT[i]*2*pi;}
  }}

  // Determine new C*(k)
  // inversion of ( P - P (H-Qs)P) for every i, stored in newh[i][j][k] (dummy tensor, just like cvec, hvec, dc, and dcHT here)

  for (i=0;i<Nr;i++) {
for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
```

```
      gsl_matrix_set (m, j, k, delta[j][k]*n[k] + n[j]*(newh[i][j][k]-QsHT[i][j][k])*n[k]);
    }}
    gsl_linalg_LU_decomp(m,p,sign);
    for (k=0;k<2*Nz;k++){
      for (j=0;j<2*Nz;j++) gsl_vector_set (cvec,j,delta[j][k]);
      gsl_linalg_LU_solve(m,p,cvec,hvec);
      for (j=0;j<2*Nz;j++) c[i][j][k] = gsl_vector_get (hvec,j);
    }
      }
      for (i=0;i<Nr;i++) {  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    c[i][j][k] -= delta[j][k]/n[j] + uRegHT[i][j][k] + QsHT[i][j][k];
    c[i][j][k] *= -1.0;
      }}}

      RecordTestfunctions();

      // Y*(k) -> Y*(r)

      for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    for (i=0;i<Nr;i++) { dc[i] = c[i][j][k]; }
    gsl_dht_apply(t, dc, dcHT); for (i=0;i<Nr;i++) dcHT[i] *= b*b/(2*pi);
    for (i=0;i<Nr;i++) { c[i][j][k] = dcHT[i];}
      }}

      cout << "Iteration " << it << "  Maximum error " << errmax << " at (i,j,k) = (" << imax << "," << jmax << "," << kmax << ") for the " << pair << " interactions" << endl;
      //  cout << "Iteration " << it << "  Maximum error " << errmax << endl;

      //  CalculateDensity(3);    RecordProfiles(false);
    }

    if (it%1 == 0 and errmax != 0) RecordCorrelationFunctions();
    if (it>50) errmax = 0;
    if (evaluations > 0 and it > 500) errmax = 0;
    if (evaluations > 0) CalculateDensity(3e4);

  } /* end of loop here */

  cout << setw(20) << iterations << " iterations after " << evaluations << " evaluations" << endl;
  if (iterations == 1 and evaluations > 30) { convergence = true; cout << "Convergence to requested precision." << endl; }
  evaluations++;

  if (errmax != 0) RecordCorrelationFunctions();

  gsl_matrix_free (m); gsl_matrix_free (m2); gsl_vector_free (hvec); gsl_vector_free (cvec); gsl_dht_free (t); gsl_permutation_free (p);
}

void CoulombFluid::AHNC_Loop(int *itn, bool InitialRuns, double lambda) {

  double I = r[Nr-1], alpha2,alphaMem,b,d;
  gsl_dht * t = gsl_dht_new(Nr, 0.0, I);
  gsl_matrix * m = gsl_matrix_alloc (2*Nz, 2*Nz),* m2 = gsl_matrix_alloc (2*Nz, 2*Nz);
  gsl_vector * hvec = gsl_vector_alloc (2*Nz), * cvec = gsl_vector_alloc (2*Nz);
  gsl_permutation * p = gsl_permutation_alloc (2*Nz);
  int sign[1], it, imax, jmax, kmax;
  char pair[3];
  bool peak = false;

  alpha2 = alphaMem = 0.5;

  b = gsl_sf_bessel_zero_J0(Nr+1)/I/I;

  it = *itn;

  errmax = 0; it++;

  /* Iteration according to Ng, J. Chem. Phys. 61, 2680 (1974), instead of Picard. Not too successful, so replaced. Present in earlier versions <= 010 */

  // new H*(r) from C*(r)
  for (i=0;i<Nr;i++) {  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    newh[i][j][k] = exp(h[i][j][k] - c[i][j][k]-u[i][j][k]+uReg[i][j][k]) -1.0;
//    newh[i][j][k] = c[i][j][k]+u[i][j][k]-uReg[i][j][k] + log(h[i][j][k]+1.0);
    if (newh[i][j][k] > 1e2 and sqrt(r[i]*r[i]+pow(z[j%Nz]-z[k%Nz],2.0)) > ap+ap) { peak = true; imax=i;jmax=j;kmax=k; }//newh[i][j][k] = c[i][j][k]+u[i][j][k]-uReg[i][j][k] + 1.0;}
  }}}
  if (peak) { cout << " ALERT high value of new total correlation function at (i,j,k) = (" << imax << "," << jmax << "," << kmax << ")" << endl; peak = false;

/*  for (i=0;i<Nr;i++) {  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    newh[i][j][k] = c[i][j][k] + u[i][j][k]-uReg[i][j][k];
  }}}*/

  }

  //  if(BulkPhase2) {
  //    for (i=0;i<Nr;i++) {  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
  //      if ( (z[Nz-1]-z[j%Nz]) < BulkCutoff or (z[Nz-1]-z[k%Nz]) < BulkCutoff ) newh[i][j][k] = newh[i][max(j-1,0)][max(k-1,0)];
  //      if ( (z[Nz-1]-z[j%Nz]) < BulkCutoff or (z[Nz-1]-z[k%Nz]) < BulkCutoff ) newh[i][j][k] = h[i][j][k];
  //    }}}
  //  }

  for (i=0;i<Nr;i++) {  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    d = sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
    if (d >=  apv[j]+apv[k]) {err = h[i][j][k]-newh[i][j][k]; if (fabs(err) > fabs(errmax)) {errmax = err;imax=i;jmax=j;kmax=k;strcpy(pair,"++");} }
    if (d >=  amv[j]+amv[k]) {err = h[i][j+Nz][k+Nz]-newh[i][j+Nz][k+Nz]; if (fabs(err) > fabs(errmax)) {errmax = err;imax=i;jmax=j;kmax=k;strcpy(pair,"--");} }
    if (d >=  apv[j]+amv[k]) {err = h[i][j][k+Nz]-newh[i][j][k+Nz]; if (fabs(err) > fabs(errmax)) {errmax = err;imax=i;jmax=j;kmax=k;strcpy(pair,"+-");} }
  }}}

  if(InitialRuns) alpha2 = min(alphaMem / fabs(errmax),0.05);  else alpha2 = alpha;
  cout << setw(20) << "alpha2 = " << alpha2 ;

  for (i=0;i<Nr;i++) {  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    d = sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
    if (d >=  apv[j]+apv[k]) {err = h[i][j][k]-newh[i][j][k]; h[i][j][k] -= alpha2*err;} else h[i][j][k] = -1.0;
    if (d >=  amv[j]+amv[k]) {err = h[i][j+Nz][k+Nz]-newh[i][j+Nz][k+Nz]; h[i][j+Nz][k+Nz] -= alpha2*err;} else h[i][j+Nz][k+Nz] = -1.0;
    if (d >=  apv[j]+amv[k]) {err = h[i][j][k+Nz]-newh[i][j][k+Nz]; h[i][j][k+Nz] -= alpha2*err; h[i][j+Nz][k] = h[i][j][k+Nz];} else h[i][j+Nz][k] = h[i][j][k+Nz] = -1.0;
  }}}
```

```
  ResetQs();

  // Determine C*(k) from (H*-Y*)(r)
  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    for (i=0;i<Nr;i++) { dc[i] = h[i][j][k]; }
    gsl_dht_apply(t, dc, dcHT);
    for (i=0;i<Nr;i++) { newh[i][j][k] = dcHT[i]*2*pi;}
  }}

  // Determine new C*(k)
  // inversion of ( P - P (H-Qs)P) for every i, stored in newh[i][j][k] (dummy tensor, just like cvec, hvec, dc, and dcHT here)

  for (i=0;i<Nr;i++) {
    for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
      gsl_matrix_set (m, j, k, delta[j][k] + (newh[i][j][k]-QsHT[i][j][k])*n[k]*lambda);
    }}
    gsl_linalg_LU_decomp(m,p,sign);
    for (k=0;k<2*Nz;k++){
      for (j=0;j<2*Nz;j++) gsl_vector_set (cvec,j,delta[j][k]);
      gsl_linalg_LU_solve(m,p,cvec,hvec);
      for (j=0;j<2*Nz;j++) newg[i][j][k] = gsl_vector_get (hvec,j);
    }
  }
  for (i=0;i<Nr;i++) {  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    newg[i][j][k] -= delta[j][k];
    newg[i][j][k] *= -1.0/n[j]/lambda;
    newg[i][j][k] += uRegHT[i][j][k] + QsHT[i][j][k];
  }}}

//  RecordTestfunctions();

  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    for (i=0;i<Nr;i++) { dc[i] = newg[i][j][k]; }
    gsl_dht_apply(t, dc, dcHT); for (i=0;i<Nr;i++) dcHT[i] *= b*b/(2*pi);
    for (i=0;i<Nr;i++) { newg[i][j][k] = dcHT[i];}
  }}

  alpha2=1.0;
  for (i=0;i<Nr;i++) {  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    {err = c[i][j][k]-newg[i][j][k]; c[i][j][k] -= alpha2*err;}
    {err = c[i][j+Nz][k+Nz]-newg[i][j+Nz][k+Nz]; c[i][j+Nz][k+Nz] -= alpha2*err;}
    {err = c[i][j][k+Nz]-newg[i][j][k+Nz]; c[i][j][k+Nz] -= alpha2*err; c[i][j+Nz][k] = c[i][j][k+Nz];}
  }}}


  cout << "\tIteration " << it << "  Maximum error " << errmax << " at (r,z,z') = (" << r[imax] << "nm," << z[jmax] << "nm," << z[kmax] << "nm) for h_(" << pair << ")" << endl;

  if (it%2 == 0 and errmax != 0) RecordCorrelationFunctions();
  if (it%2 == 0 and errmax != 0) RecordTestfunctions();
  if (it>50) { errmax = 0; cout << "Warning, iteration is broken off here" << endl; } //WARNING iteration is broken off here
//  if (it>50) errmax = 0;
//  if (evaluations > 0 and it > 500) errmax = 0;
//  if (evaluations > 0) CalculateDensity(3e4);

  *itn = it;

  gsl_matrix_free (m); gsl_matrix_free (m2); gsl_vector_free (hvec); gsl_vector_free (cvec); gsl_dht_free (t); gsl_permutation_free (p);
}

void CoulombFluid::AHNC_convolute_Loop(int *itn, bool InitialRuns, double lambda) {

  double I = r[Nr-1], alpha2,alphaMem,b,d, *ThetaConv, thetaMax, A;
  gsl_dht * t = gsl_dht_new(Nr, 0.0, I);
  gsl_matrix * m = gsl_matrix_alloc (2*Nz, 2*Nz),* m2 = gsl_matrix_alloc (2*Nz, 2*Nz);
  gsl_vector * hvec = gsl_vector_alloc (2*Nz), * cvec = gsl_vector_alloc (2*Nz);
  gsl_permutation * p = gsl_permutation_alloc (2*Nz);
  int sign[1], it, imax, l;
  char pair[3],answer;
  bool peak = false, DontAskAgain=false;

  ThetaConv = (double*) calloc (Nr,sizeof(double));

  alpha2 = alphaMem = 0.5;

  b = gsl_sf_bessel_zero_J0(Nr+1)/I/I;

  it = *itn;

  errmax = 0; it++;

  /* Iteration according to Ng, J. Chem. Phys. 61, 2680 (1974), instead of Picard. Not too successful, so replaced. Present in earlier versions <= 010 */

  // new H*(r) from C*(r)
  i=0; while (r[i] <= 5.0/kappa1+2.0*ap) i++; l=i;
  for (i=0;i<Nr;i++) {  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    A = h[i][j][k] - c[i][j][k]-(u[i][j][k]-uReg[i][j][k])*lambda;
    if (fabs(A) < 10e-2) { newh[i][j][k] = A + A*A/2.0 + A*A*A/6.0 + A*A*A*A/24.0; } else { newh[i][j][k] = exp(A) -1.0; }
    //    newh[i][j][k] = c[i][j][k]+u[i][j][k]-uReg[i][j][k] + log(h[i][j][k]+1.0);
    if (newh[i][j][k] > 1.0e2 and sqrt(r[i]*r[i]+pow(z[j%Nz]-z[k%Nz],2.0)) > ap+ap) { peak = true; imax=i;jmax=j;kmax=k; }//newh[i][j][k] = c[i][j][k]+u[i][j][k]-uReg[i][j][k] + 1.0;}
    //if (i>l) newh[i][j][k] = newh[l][j][k]*exp(-kappa1*qp*(r[i]-r[l]));//newh[l][j][k]*exp((newh[l][j][k]-newh[l-1][j][k])/(r[l]-r[l-1])*(r[i]-r[l])/newh[l][j][k]);
  }}}
  if (peak) { cout << " ALERT high value of new total correlation function at (i,j,k) = (" << imax << "," << jmax << "," << kmax << ")" << endl; peak = false; }

//  if(BulkPhase2) {
//    for (i=0;i<Nr;i++) {  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
//      if ( (z[Nz-1]-z[j%Nz]) < BulkCutoff or (z[Nz-1]-z[k%Nz]) < BulkCutoff ) newh[i][j][k] = newh[i][max(j-1,0)][max(k-1,0)];
//      if ( (z[Nz-1]-z[j%Nz]) < BulkCutoff or (z[Nz-1]-z[k%Nz]) < BulkCutoff ) newh[i][j][k] = h[i][j][k];
//    }}}
//  }

  for (i=0;i<Nr;i++) {  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    d = sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0))*0.5; //ALERT;
    if (d >=  apv[j]+apv[k]) {err = h[i][j][k]-newh[i][j][k]; if (fabs(err) > fabs(errmax)) {errmax = err;imax=i;jmax=j;kmax=k;strcpy(pair,"++");} }
    if (d >=  amv[j]+amv[k]) {err = h[i][j+Nz][k+Nz]-newh[i][j+Nz][k+Nz]; if (fabs(err) > fabs(errmax)) {errmax = err;imax=i;jmax=j+Nz;kmax=k+Nz;strcpy(pair,"--");} }
```

```
    if (d >=  apv[j]+amv[k]) {err = h[i][j][k+Nz]-newh[i][j][k+Nz]; if (fabs(err) > fabs(errmax)) {errmax = err;imax=i;jmax=j;kmax=k+Nz;strcpy(pair,"+-");} }
    if (d >=  apv[k]+amv[j]) {err = h[i][j+Nz][k]-newh[i][j+Nz][k]; if (fabs(err) > fabs(errmax)) {errmax = err;imax=i;jmax=j+Nz;kmax=k;strcpy(pair,"-+");} }
  }}}

  if(InitialRuns) alpha2 = min(alphaMem / fabs(errmax),alphaMin);  else alpha2 = alpha;
  cout << setw(20) << "alpha2 = " << alpha2 ;

//  if (fabs(errmax) > fabs(errMem[0])) alphaMin /= 1.5;
//  dummy=0; for (i=1;i<Nerr-1;i++) dummy += pow( ((errMem[Nerr-1]-errMem[0])/Nerr*i +errMem[0]-errMem[i]) / errMem[i] , 2.0);
//  if (dummy<errLimitSq) alphaMin *= 1.1;
//  for (i=1;i<Nerr;i++) errMem[i]=errMem[i-1];

  errMem[0] = errmax;

  for (i=0;i<Nr;i++) {  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    d = sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
    if (d >=  apv[j]+apv[k]) {err = h[i][j][k]-newh[i][j][k]; h[i][j][k] -= alpha2*err;} else h[i][j][k] = -1.0;
    if (d >=  amv[j]+amv[k]) {err = h[i][j+Nz][k+Nz]-newh[i][j+Nz][k+Nz]; h[i][j+Nz][k+Nz] -= alpha2*err;} else h[i][j+Nz][k+Nz] = -1.0;
    if (d >=  apv[j]+amv[k]) {err = h[i][j][k+Nz]-newh[i][j][k+Nz]; h[i][j][k+Nz] -= alpha2*err; } else h[i][j][k+Nz] = -1.0;
    if (d >=  apv[k]+amv[j]) {err = h[i][j+Nz][k]-newh[i][j+Nz][k]; h[i][j+Nz][k] -= alpha2*err; } else h[i][j+Nz][k] = -1.0;
  }}}


  ResetQs();
  /*
  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    for (l=1;l<Nr;l++) {
      newg[l][j][k] = 0;
      for (i=1;i<Nr;i++) {
    d =  period*period - pow(r[i]-r[l],2.0);
    if ( period*period - pow(r[i]+r[l],2.0) > 0 ) thetaMax = pi; else thetaMax = acos( 1.0 - d/(2.0*r[i]*r[l]) );
        if (d>0) newg[l][j][k] += (r[i]-r[i-1])*r[i]*h[i][j][k]*thetaMax;
      }
    }
    newg[0][j][k] = newg[1][j][k];
  }}

  if (it<200) { gsl_dht_apply(t, Theta, ThetaConv); for (i=0;i<Nr;i++) ThetaConv[i] *= b*b/(2*pi); RecordConvolutionFunctions(ThetaConv); }
  */
  // Determine C*(k) from (H*-Y*)(r)
  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    for (i=0;i<Nr;i++) { dc[i] = h[i][j][k]; }
    gsl_dht_apply(t, dc, dcHT);
    for (i=0;i<Nr;i++) { newh[i][j][k] = dcHT[i]*2*pi;}
  }}

  // Determine new C*(k)
  // inversion of ( P - P (H-Qs)P) for every i, stored in newh[i][j][k] (dummy tensor, just like cvec, hvec, dc, and dcHT here)

  for (i=0;i<Nr;i++) {
    for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
      gsl_matrix_set (m, j, k, delta[j][k]*n[k] + (newh[i][j][k]-QsHT[i][j][k])*n[j]*n[k]);
    }}
    gsl_linalg_LU_decomp(m,p,sign);
    for (k=0;k<2*Nz;k++){
      for (j=0;j<2*Nz;j++) gsl_vector_set (cvec,j,delta[j][k]);
      gsl_linalg_LU_solve(m,p,cvec,hvec);
      for (j=0;j<2*Nz;j++) c[i][j][k] = gsl_vector_get (hvec,j);
    }
  }


  for (i=0;i<Nr;i++) {  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    c[i][j][k] -= delta[j][k]/n[k];
    c[i][j][k] *= -1.0;
    c[i][j][k] += uRegHT[i][j][k]*lambda + QsHT[i][j][k];
  }}}

  if(it<0) {
    for (i=0;i<Nr;i++) { for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
      if(pow(z[j%Nz]-z[k%Nz],2.0) + r[i]*r[i]>4*ap*ap) c[i][j][k] *= exp(-2*kappa1*sqrt(pow(z[j%Nz]-z[k%Nz],2.0) + r[i]*r[i]));
    }}}
    cout << "correction to c(k=0) is made" << endl;
  }
  //if (!DontAskAgain) { cout << "Break to test correlation functions (s = stop making breaks)" << endl; answer = cin.get (); if (answer == 's') DontAskAgain=true; }

  RecordCorrelationFunctions();
  //Alternative calculation of C(k)
  /*
  for (i=0;i<Nr;i++) {  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    c[i][j][k] *= newh[i][j][k]-QsHT[i][j][k];
    //c[i][j][k] *= -1.0/n[j];
    c[i][j][k] += uRegHT[i][j][k]*lambda + QsHT[i][j][k];
  }}}
  */

  //  RecordTestfunctions();

  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    for (i=0;i<Nr;i++) { dc[i] = c[i][j][k]; }
    gsl_dht_apply(t, dc, dcHT); for (i=0;i<Nr;i++) dcHT[i] *= b*b/(2*pi);
    for (i=0;i<Nr;i++) { c[i][j][k] = dcHT[i]; }//     if (i>Nr/2.0) c[i][j][k] = uReg[i][j][k]-u[i][j][k]; } //ALERt
  }}

  // evaluate a second C* by the HNC closure, and mix with the other solution for C*
  /*
  for (i=0;i<Nr;i++) {  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    d = sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
    if (d >=  apv[j]+apv[k]) newh[i][j][k] = -(u[i][j][k]-uReg[i][j][k])*lambda + h[i][j][k] - log(h[i][j][k]+1.0);
    if (d >=  amv[j]+amv[k]) newh[i][j+Nz][k+Nz] = -(u[i][j+Nz][k+Nz]-uReg[i][j+Nz][k+Nz])*lambda + h[i][j+Nz][k+Nz] - log(h[i][j+Nz][k+Nz]+1.0);
    if (d >=  apv[j]+amv[k]) { newh[i][j][k+Nz] = -(u[i][j][k+Nz]-uReg[i][j][k+Nz])*lambda + h[i][j][k+Nz] - log(h[i][j][k+Nz]+1.0);
    newh[i][j+Nz][k]=newh[i][j][k+Nz]; }
  }}}

  for (i=0;i<Nr;i++) {  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
```

```
    d = sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
    if (d >=  apv[j]+apv[k]) c[i][j][k] = (1.0-alpha2)*newh[i][j][k] + alpha2*c[i][j][k];
    if (d >=  amv[j]+amv[k]) c[i][j+Nz][k+Nz] = (1.0-alpha2)*newh[i][j+Nz][k+Nz] + alpha2*c[i][j+Nz][k+Nz];
    if (d >=  apv[j]+amv[k]) { c[i][j][k+Nz] = (1.0-alpha2)*newh[i][j][k+Nz] + alpha2*c[i][j][k+Nz]; c[i][j+Nz][k]=c[i][j][k+Nz]; }
  }}}
  */

  cout << "\tIteration " << it << "  Maximum error " << errmax << " at (r,z,z') = (" << r[imax] << "nm," << z[jmax%Nz] << "nm," << z[kmax%Nz] << "nm) for h_(" << pair << ")" << endl;

  if (it%2 == 0 and errmax != 0) RecordCorrelationFunctions();
  if (it%2 == 0 and errmax != 0) RecordTestfunctions();
  //if (it%50==49) { errmax = 1e-3; cout << "Warning, iteration is broken off here" << endl; } //WARNING iteration is broken off here
  //  if (it>50) errmax = 0;
  //  if (evaluations > 0 and it > 500) errmax = 0;
  //  if (evaluations > 0) CalculateDensity(3e4);

  *itn = it;

  //GnuplotCorrelationFunctions();

  gsl_matrix_free (m); gsl_matrix_free (m2); gsl_vector_free (hvec); gsl_vector_free (cvec); gsl_dht_free (t); gsl_permutation_free (p); free(ThetaConv);
}

void CoulombFluid::AMHNC_Loop(int *itn, bool InitialRuns, double lambda) {

  double I = r[Nr-1], alpha2,alphaMem,b,d, *ThetaConv, thetaMax;
  gsl_dht * t = gsl_dht_new(Nr, 0.0, I);
  gsl_matrix * m = gsl_matrix_alloc (2*Nz, 2*Nz),* m2 = gsl_matrix_alloc (2*Nz, 2*Nz);
  gsl_vector * hvec = gsl_vector_alloc (2*Nz), * cvec = gsl_vector_alloc (2*Nz);
  gsl_permutation * p = gsl_permutation_alloc (2*Nz);
  int sign[1], it, imax;
  char pair[3];

  ThetaConv = (double*) calloc (Nr,sizeof(double));

  alpha2 = alphaMem = 315.5;

  b = gsl_sf_bessel_zero_J0(Nr+1)/I/I;

  it = *itn;

  errmax = 0; it++;

  /* Iteration according to Ng, J. Chem. Phys. 61, 2680 (1974), instead of Picard. Not too successful, so replaced. Present in earlier versions <= 010 */

  // new H*(r) from C*(r)
  for (i=0;i<Nr;i++) {  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    d = sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
    if (d >=  apv[j]+apv[k]) newh[i][j][k] = -u[i][j][k]+0*uReg[i][j][k];// + h[i][j][k] - log(h[i][j][k]+1.0);
    if (d >=  amv[j]+amv[k]) newh[i][j+Nz][k+Nz] = -u[i][j+Nz][k+Nz]+0*uReg[i][j+Nz][k+Nz];// + h[i][j+Nz][k+Nz] - log(h[i][j+Nz][k+Nz]+1.0);
    if (d >=  apv[j]+amv[k]) { newh[i][j][k+Nz] = -u[i][j][k+Nz]+0*uReg[i][j][k+Nz];// + h[i][j][k+Nz] - log(h[i][j][k+Nz]+1.0);
    newh[i][j+Nz][k]=newh[i][j][k+Nz]; }
  }}}

  for (i=0;i<Nr;i++) {  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    err = c[i][j][k]-newh[i][j][k]; if (fabs(err) > fabs(errmax)) {errmax = err;imax=i;jmax=j;kmax=k;strcpy(pair,"++");}
    err = c[i][j+Nz][k+Nz]-newh[i][j+Nz][k+Nz]; if (fabs(err) > fabs(errmax)) {errmax = err;imax=i;jmax=j+Nz;kmax=k+Nz;strcpy(pair,"--");}
    err = c[i][j][k+Nz]-newh[i][j][k+Nz]; if (fabs(err) > fabs(errmax)) {errmax = err;imax=i;jmax=j;kmax=k+Nz;strcpy(pair,"+-");}
  }}}

  if(InitialRuns) alpha2 = min(alphaMem / fabs(errmax),0.5);  else alpha2 = alpha;
  cout << setw(20) << "alpha2 = " << alpha2 ;

  for (i=0;i<Nr;i++) {  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    err = c[i][j][k]-newh[i][j][k]; c[i][j][k] -= alpha2*err;
    err = c[i][j+Nz][k+Nz]-newh[i][j+Nz][k+Nz]; c[i][j+Nz][k+Nz] -= alpha2*err;
    err = c[i][j][k+Nz]-newh[i][j][k+Nz]; c[i][j][k+Nz] -= alpha2*err; c[i][j+Nz][k] = c[i][j][k+Nz];
  }}}

  // Determine C*(k) from C*(r)
  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    for (i=0;i<Nr;i++) { dc[i] = c[i][j][k]; }
    gsl_dht_apply(t, dc, dcHT);
    for (i=0;i<Nr;i++) { newh[i][j][k] = dcHT[i]*2*pi;}
  }}

  // Determine new H*(k)
  // inversion of ( I - P C ) for every i, stored in newh[i][j][k] (dummy tensor, just like cvec, hvec, dc, and dcHT here)

  for (i=0;i<Nr;i++) {
    for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
      gsl_matrix_set (m, j, k, delta[j][k] - (newh[i][j][k]-0*uReg[i][j][k])*n[k]);
    }}
    gsl_linalg_LU_decomp(m,p,sign);
    for (k=0;k<2*Nz;k++){
      for (j=0;j<2*Nz;j++) gsl_vector_set (cvec,j,delta[j][k]);
      gsl_linalg_LU_solve(m,p,cvec,hvec);
      for (j=0;j<2*Nz;j++) h[i][j][k] = gsl_vector_get (hvec,j);
    }
  }
  for (i=0;i<Nr;i++) {  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    h[i][j][k] -= delta[j][k];
    h[i][j][k] /= n[j];
  }}}

  //  RecordTestfunctions();

  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    for (i=0;i<Nr;i++) { dc[i] = h[i][j][k]; }
    gsl_dht_apply(t, dc, dcHT); for (i=0;i<Nr;i++) dcHT[i] *= b*b/(2*pi);
    for (i=0;i<Nr;i++) { h[i][j][k] = dcHT[i];}
  }}
  //for (i=Nr/2;i<Nr;i++) {  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
  //  h[i][j][k] = h[Nr/2][j][k]*exp(-kappa1*qp*(r[i]-r[Nr/2])); //ALERt
  //}}}
```

```
  // Calculate C* again, to get a new estimate for the behavior for r < hard core

  for (i=0;i<Nr;i++) {  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    d = sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
    if (d <  apv[j]+apv[k]) h[i][j][k] = -1.0;
    if (d <  amv[j]+amv[k]) h[i][j+Nz][k+Nz] = -1.0;
    if (d <  apv[j]+amv[k]) { h[i][j][k+Nz] = -1.0; h[i][j+Nz][k]=h[i][j][k+Nz]; }
  }}}


  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    for (i=0;i<Nr;i++) { dc[i] = h[i][j][k]; }
    gsl_dht_apply(t, dc, dcHT);
    for (i=0;i<Nr;i++) { newh[i][j][k] = dcHT[i]*2*pi;}
  }}

  // Determine new C*(k)
  // inversion of ( P - P (H-Qs)P) for every i, stored in newh[i][j][k] (dummy tensor, just like cvec, hvec, dc, and dcHT here)

  for (i=0;i<Nr/2;i++) {
    for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
      gsl_matrix_set (m, j, k, delta[j][k] + newh[i][j][k]*n[k]);
    }}
    gsl_linalg_LU_decomp(m,p,sign);
    for (k=0;k<2*Nz;k++){
      for (j=0;j<2*Nz;j++) gsl_vector_set (cvec,j,delta[j][k]);
      gsl_linalg_LU_solve(m,p,cvec,hvec);
      for (j=0;j<2*Nz;j++) newh[i][j][k] = gsl_vector_get (hvec,j);
    }
  }
  for (i=0;i<Nr;i++) {  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    newh[i][j][k] -= delta[j][k];
    newh[i][j][k] *= -1.0/n[j]/lambda;
    newh[i][j][k] += 0*uRegHT[i][j][k];
  }}}

  //  RecordTestfunctions();

  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    for (i=0;i<Nr;i++) { dc[i] = newh[i][j][k]; }
    gsl_dht_apply(t, dc, dcHT); for (i=0;i<Nr;i++) dcHT[i] *= b*b/(2*pi);
    for (i=0;i<Nr;i++) { newh[i][j][k] = dcHT[i];}
  }}



  cout << "\tIteration " << it << "  Maximum error " << errmax << " at (r,z,z') = (" << r[imax] << "nm," << z[jmax%Nz] << "nm," << z[kmax%Nz] << "nm) for c_(" << pair << ")" << endl;

  if (it%2 == 0 and errmax != 0) RecordCorrelationFunctions();
  if (it%2 == 0 and errmax != 0) RecordTestfunctions();
//  if (it>50) errmax = 0;
//  if (evaluations > 0 and it > 500) errmax = 0;
//  if (evaluations > 0) CalculateDensity(3e4);

  *itn = it;

  gsl_matrix_free (m); gsl_matrix_free (m2); gsl_vector_free (hvec); gsl_vector_free (cvec); gsl_dht_free (t); gsl_permutation_free (p); free(ThetaConv);
}

void CoulombFluid::OZ_AHNC_AdaptiveFinder(double precision) {

  double Error, minError,alphamem, lambda=1.0, *nMem;
  int it=0, trial, trialmax=3, runs=0, maxruns= 1000000;
  bool InitialRuns = true;

  nMem = (double*) calloc (2*Nz,sizeof(double));
  for (j=0;j<2*Nz;j++) { nMem[j] = n[j]; }
  //if (evaluations==0) for (j=0;j<Nz;j++) { nOld[j]=3/2*cs1*dz[j]; nOld[j+Nz]=3/2*qp/qm*cs1*dz[j]; }

  ResetQs();
  // if (evaluations>2) for (j=0;j<2*Nz;j++) { n[j] = (n[j]+nOld[j])/2.0; nOld[j]=n[j]; } /* ALERT; Careful with this operation - often not desirable! */
  //  else for (j=0;j<2*Nz;j++) nOld[j]=n[j];

  //if (evaluations>=0 and evaluations < 6) for (j=0;j<2*Nz;j++) { n[j] = 0.1*n[j]+0.9*nOld[j]; nOld[j]=n[j]; } /* ALERT; Careful with this operation - often not desirable! ... but sometimes necessary */

  //if (evaluations>90) for (j=0;j<2*Nz;j++) { n[j] = 0.9*n[j]+0.1*nOld[j]; nOld[j]=n[j]; } /* ALERT; Careful with this operation - often not desirable! ... but sometimes necessary */
  //else for (j=0;j<2*Nz;j++) nOld[j]=n[j];

  RecordCorrelationFunctions();

  errmax = 1e9;

  cout << "\n** Performing the initial runs with adaptive alpha, for " << maxruns << " runs **\n" << endl;

  alphamem=alpha; if (evaluations < 0) lambda = 0.1;
  while ((fabs(errmax) > precision and runs < maxruns) or (lambda < 1.0)) {
    Error = errmax;
//    AMHNC_Loop(&it,InitialRuns,lambda);
    AHNC_convolute_Loop(&it,InitialRuns,lambda);
//    if (Error < errmax and runs > 30) { errmax = precision; cout << "ALERT: iteration interrupted by increase of error" << endl; }
//    if (Error < errmax and runs > 30) { cout << "ALERT: iteration interrupted by increase of error" << endl; break; }
//    if (Error < errmax and runs > 5) { CalculateDensity(10); cout << "ALERT: iteration interrupted by increase of error" << endl; }
    runs++;
    if (lambda < 1.0 and fabs(errmax) < precision*2) { lambda += 0.01; cout << "lambda = " << lambda << endl; }
//    CalculateDensity(0);
    if (runs < 10) errmax = 1e200;
    if (evaluations > 30 and fabs(errmax)<precision/10.0) { errmax = precision/1e2; cout << "Alert: possible convergence, not certain" << endl; }
    if (evaluations > 40 ) { errmax = precision/1e2; cout << "Alert: possible convergence, not certain" << endl; }
  }
  /* end of initial loops here */

  if (InitialRuns and fabs(errmax) > precision) {
    InitialRuns = false;
    cout << "\n** Initial runs have been performed. Next runs will use trial values for the alpha parameter **\n" << endl;
  }
```

```
  alpha0[0]=alpha;
  while (fabs(errmax) > precision or lambda < 1.0) {

    InitialCorrelationFunctions(); minError=10e200;

    for (trial=0;trial<trialmax;trial++) {

      SetCorrelationFunctions();
      alpha = alpha0[0]*pow(5.0,-(trialmax-1.0)/2.0 + trial)*pow(1.0,trial*1.0);
      if (alpha > 1) continue;

      AHNC_Loop(&it,InitialRuns,lambda);      if (lambda < 1.0) { lambda += 0.01; cout << "lambda = " << lambda << endl; }

      AHNC_Loop(&it,InitialRuns,lambda);      if (lambda < 1.0) { lambda += 0.01; cout << "lambda = " << lambda << endl; }

      Error = fabs(errmax); cout << "Last error of trial " << trial << " with alpha = " << alpha << " had a maximum value of " << errmax << endl;

      if (Error < minError) { minError=Error; FinalCorrelationFunctions(); }

    }
    GetCorrelationFunctions();

    cout << "\nOptimal alpha was found for alpha = " << alpha0[0] << " with maximum error " << errmax << "\n" << endl;

    if (alpha < 1e-12) { cout << "iteration break because of small alpha parameter" << endl; break; }

  } /* end of loop here */
  alpha=alphamem;


  cout << setw(20) << it << " iterations after " << evaluations << " evaluations" << endl;
  if (iterations == 1 and evaluations > 30) { convergence = true; cout << "Convergence to requested precision." << endl; }
  evaluations++;

  if (errmax != 0) RecordCorrelationFunctions();
  if (fabs(errmax) < precision*1e-1) convergence = true;

  //for (j=0;j<2*Nz;j++) { n[j] = nMem[j]; }

  free(nMem);

}

void CoulombFluid::CalculateDensity(int Nmax) {

  double *newcp, *newcm, norm1, norm2;
  int SCount=0, jmax=0;
  bool LowDens=false;
  char type[40];


  iterations=0;

  newcp = (double*) calloc (Nz,sizeof(double));  newcm = (double*) calloc (Nz,sizeof(double));

for (j=0;j<2*Nz;j++) { for (k=0;k<2*Nz;k++) {
  mu[j][k] = 0;
}}

/* h[i][j][k] is the discretized h_ij(r)+Qs_ij(r), with Qs the 'smoothing function'. */
if (correlations) {
  for (j=0;j<2*Nz;j++) { for (k=0;k<2*Nz;k++) {
    for (i=1;i<Nr;i++) {
  mu[j][k] += pi*(r[i]*r[i]-r[i-1]*r[i-1])*(0.5*(h[i][j][k]-Qs[i][j][k])*(h[i][j][k]-c[i][j][k]+uReg[i][j][k]) - c[i][j][k] + Qs[i][j][k] + uReg[i][j][k] - u[i][j][k]);
    }
    //      mu[k][j] = mu[j][k];
  }}
  for (j=0;j<2*Nz;j++) { for (k=0;k<2*Nz;k++) {
    //      newMu[j][k] = (1.0-alpha)*newMu[j][k] + alpha*mu[j][k];
    newMu[j][k] = mu[j][k];
  }}
} /* ALERT check mu - &has been checked many times - and changed recently - &checked */
else {
  for (i=0;i<Nr;i++) {  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    c[i][j][k] = uReg[i][j][k]-u[i][j][k];
    c[i][j][k] *= 0.8;
  }}}
  for (j=0;j<2*Nz;j++){
    muexp[j] = muexm[j] = 0.0;
  }
}

errmax=1.0;

  if(correlations) {
    for (j=0;j<Nz;j++) {
  norm1=norm2=0;
  for (k=0;k<2*Nz;k++) norm1 += n[k]*newMu[j][k];
  muexp[j] = norm1;
  for (k=0;k<2*Nz;k++) norm2 += n[k]*newMu[j+Nz][k];
  muexm[j] = norm2;
    }
  //      muEx1p = muexp[Nz-1]; muEx1m = muexm[Nz-1];
  } //RecordProfiles(false);
while (errmax > tolerance) {
  errmax = 0.0;
  //CalculatePotentialWithTraceSalt();
      CalculateEfieldAndPotential();
  //  CalculateMeanPotential1boundary();

  //    CalculateExpensivePotential2Walls();

  //    CalculateExpensivePotential(); //ALERt phi is not evaluated
```

```
if(iterations%1000000 == -1)  Potentiostat();

/* the Boltzmann distributions */
for (j=0;j<Nz;j++) {
  //    norm1 = newcp[j]*4*pi*pow(ap,3.0)/3.0 + newcm[j]*4*pi*pow(am,3.0)/3.0;
  //newcp[j] = cs1*exp(-qpv[j]*phi[j] - Vextp[j] - muexp[j]);// - newcp[j]*(4*norm1-3*norm1*norm1)/pow(1.0-norm1,2.0));
  newcp[j] =       cs1*exp(-qp*phi[j] - Vextp[j] - muexp[j] + muEx1p);
  newcm[j] = qp/qm*cs1*exp( qm*phi[j] - Vextm[j] - muexm[j] + muEx1m);
  //newcm[j] = qp/qm*cs1*exp( qmv[j]*phi[j] - Vextm[j] - muexm[j]);// - newcm[j]*(4*norm1-3*norm1*norm1)/pow(1.0-norm1,2.0)); /* ALERt : phi is gone, and -u in mu*/
}
//  newcp[0] = 1e-30/dz[0];  newcm[0] = 0.01/dz
  norm1 = norm2 = 0.0;
  norm2 = cs1*3.0; //ALERT
//for (j=0;j<Nz;j++) { norm1 += newcp[j]*dz[j]; } //norm2 += cs1*dz[j]; } /* For canonical ensemble */
//for (j=0;j<Nz;j++) { newcp[j] *= norm2/norm1; }
//norm1 = 0.0;
//for (j=0;j<Nz;j++) { norm1 += newcm[j]*dz[j];}
//for (j=0;j<Nz;j++) { newcm[j] *= qp/qm*norm2/norm1; }


/*  norm1 = newcp[Nz/2];  norm2 = newcm[Nz/2];
for (j=0;j<Nz;j++) if (z[j] >=0 ) { newcp[j] *= cs2/norm1; }
for (j=0;j<Nz;j++) if (z[j] >=0 ) { newcm[j] *= qp/qm*cs2/norm2; }
*/
if (BulkPhase2) {
  for (j=0;j<Nz;j++) { if ( (z[Nz-1]-z[j%Nz]) < BulkCutoff ) { newcp[j] = newcp[j-1]; newcm[j] = newcm[j-1]; } }
  norm1 = newcp[Nz-1];    norm2 = newcm[Nz-1];
  for (j=0;j<Nz;j++) { newcp[j] *= cs1/norm1; newcm[j] *= qp/qm*cs1/norm2; }
}

/*  norm1 = newcp[0];  norm2 = newcm[0];
for (j=0;j<Nz;j++) { newcm[j] *= qp/qm*norm1/norm2; }*/

/* error check, and new solution */
for (j=0;j<Nz;j++) {
  //err = (cp[j]-newcp[j])/cp[j]; if(errmax<fabs(err)) { errmax = fabs(err); jmax=j; strcpy(type,"cations"); }
  err = (cp[j]-newcp[j]); if(errmax<=fabs(err)) { errmax = fabs(err); jmax=j; strcpy(type,"cations"); }
  cp[j] -= alpha*err*cp[j];
  //err = (cm[j]-newcm[j])/cm[j]; if(errmax<fabs(err)) { errmax = fabs(err); jmax=j; strcpy(type,"anions"); }
  err = (cm[j]-newcm[j]); if(errmax<=fabs(err)) { errmax = fabs(err); jmax=j; strcpy(type,"anions"); }
  cm[j] -= alpha*err*cm[j];
}
for (j=0;j<Nz;j++)    { n[j] = cp[j]*dz[j]; }    for (j=Nz;j<2*Nz;j++) { n[j] = cm[j-Nz]*dz[j-Nz]; }
//for (j=0;j<Nz;j++)  { if (Vextp[j] < 0.0) n[j] = cp[j]*dz[Nz-1]; else n[j] = cp[j]*dz[Nz/2];} /* ALERt careful: option to remove grid artifacts, use with care.  */
//for (j=Nz;j<2*Nz;j++) { if (Vextm[j] < 0.0) n[j] = cm[j-Nz]*dz[Nz-1];else n[j] = cm[j-Nz]*dz[Nz/2];} /* ALERt careful: option to remove grid artifacts, use with care.  */
//for (j=0;j<Nz;j++)  { n[Nz+j]=n[j]; } // ALERt this is only valid in symmetric situations (ions are of equal size and of equal absolute charge)

if (!correlations) for(j=0;j<Nz;j++) { cpMF[j]=cp[j]; cmMF[j]=cm[j]; }
if (iterations%10000 == 0)  {
  cout << setw(20) << iterations << " iterations;\tmaximal error is" << setw(14) << errmax << " at z = " << z[jmax] << " nm for the " << type << endl;
}
if (iterations > Nmax) break; /* ALERT Iteration is broken off here */
  for (j=1;j<Nz-1;j++) {
    //cout << setw(15) << "z" << setw(15) << z[j] << setw(15) << "n" << setw(15) << n[j] << setw(15) << "cp1" << setw(15) << cp[j] << setw(15) << "cm1" << setw(15) << cm[j] << endl;
  }

  //if(evaluations >= 0 and iterations%10000 == 0) GnuplotDensity();
  if(iterations%100000 == 0) GnuplotDensity();


  iterations++;

  } /* iteration ends here */
  cout << setw(20) << iterations << " iterations;\tmaximal error is" << setw(14) << errmax << " at z = " << z[jmax] << " nm for the " << type << endl;

  for (j=0;j<Nz;j++)    { n[j] = cp[j]*dz[j]; }
  for (j=Nz;j<2*Nz;j++) { n[j] = cm[j-Nz]*dz[j-Nz]; }
  //for (j=0;j<Nz;j++)  { n[Nz+j]=n[j]; } // ALERt this is only valid in symmetric situations (ions are of equal size and of equal absolute charge)
  for (j=0;j<Nz;j++)    { if (muexm[j] != muexm[j]) SCount++;  if (muexp[j] != muexp[j]) SCount++; }

  //for (j=0;j<Nz;j++)      { if (n[j] < 1e-1*cs1*dz[j]) { n[j] = 1e-1*cs1*dz[j]; LowDens = true; } }
  //for (j=Nz;j<2*Nz;j++)   { if (n[j] < 1e-1*cs1*dz[j]*qp/qm) { n[j] = 1e-1*cs1*dz[j]*qp/qm; LowDens = true; } }
  if (LowDens) { cout << "Adapted density" << endl; LowDens=false; }
  if (SCount != 0) { cout << SCount << " singularities were counted" << endl; Singularities = true; return; }
  //  else RecordProfiles(false);

  /*for (i=0;i<Nr;i++) {  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    d = sqrt(r[i]*r[i] + pow(z[j]-z[k],2.0));
    if (d >=  apv[j]+apv[k]) {h[i][j][k]=0.0;c[i][j][k]/=0.0;}
    if (d >=  amv[j]+amv[k]) {h[i][j+Nz][k+Nz]=0.0; c[i][j+Nz][k+Nz]=0.0;}
    if (d >=  apv[j]+amv[k]) {h[i][j][k+Nz]=0.0; c[i][j][k+Nz]=0.0; h[i][j+Nz][k] = h[i][j][k+Nz]; c[i][j+Nz][k] = c[i][j][k+Nz];}
  }}}*/




  free(newcp); free(newcm);

}

void CoulombFluid::CalculateMeanPotentialSlit() {

  double A1,sum;
  double * Q;

  Q = (double*) calloc (Nz,sizeof(double));

  for (j=0;j<Nz;j++) { Q[j] = qpv[j]*cp[j] - qmv[j]*cm[j];}

  A1 = 2*pi*lB1;

  for (j=1;j<Nz-1;j++) {
    sum = 0;
```

```
    for (k=1;k<Nz-1;k++) {
      sum += A1*(fabs(z[k]-z[j]))*  Q[k] *dz[k];
    }
    phi[j] = -sum;
//    if (j>1) phi[j] -= phi[1];
  }
  for (j=1;j<Nz-1;j++) {
    if (j!= Nz/2) phi[j] -= phi[Nz/2];
  } phi[Nz/2]=0;
  phi[0] = phi[1]; phi[Nz-1] = phi[Nz-2];

  free(Q);

}

void CoulombFluid::CalculateMeanPotential1boundary() {

  double norm, *newphi;
  gsl_vector * Q = gsl_vector_alloc (Nz-2),* diag = gsl_vector_alloc (Nz-2), * up = gsl_vector_alloc (Nz-3), * low = gsl_vector_alloc (Nz-3), * phivec = gsl_vector_alloc (Nz-2);

  newphi = (double*) calloc (Nz,sizeof(double));
  interface = Nz/2; // ALERT added
  for (j=0;j<interface-2;j++) {     gsl_vector_set(Q,j,-4*pi*lB1*(qp*cp[j+1] - qm*cm[j+1]));  }
  for (j=interface-1;j<Nz-2;j++) {  gsl_vector_set(Q,j,-4*pi*lB2*(qp*cp[j+1] - qm*cm[j+1]));  }
  for (j=0;j<Nz-2;j++) {
    norm = (z[j+2]-z[j+1])*(z[j+1]-z[j])*(z[j+2]-z[j])*0.5;
    gsl_vector_set(diag,j,-(z[j+2]-z[j])/norm);
    if (j == Nz-3) break;
    gsl_vector_set(up,j,(z[j+1]-z[j])/norm);
    gsl_vector_set(low,j,(z[j+3]-z[j+2])/norm);
  }
/*  // boundary condition at interface (continuous displacement field)
    norm = (z[interface]+z[interface-1]-2*z[interface-2])*(z[interface-1]-z[interface-2])*(z[interface]-z[interface-1])*0.125;
    gsl_vector_set(diag,interface-2,-(z[interface-1]-z[interface-2])*e2/(e1+e2)/norm - 0.5*(z[interface]-z[interface-1])/norm);
    gsl_vector_set(up,interface-2,(z[interface-1]-z[interface-2])*e2/(e1+e2)/norm);// + 0*pow(z[interface]-z[interface-2],-2.0)*(lB1/lB2-1.0) ); // last term would be due to the jump in permittivity
    gsl_vector_set(low,interface-3, 0.5*(z[interface]-z[interface-1])/norm);//    - 0*pow(z[interface]-z[interface-2],-2.0)*(lB1/lB2-1.0) );

    norm = (2*z[interface+1]-z[interface]-z[interface-1])*(z[interface]-z[interface-1])*(z[interface+1]-z[interface])*0.125;
    gsl_vector_set(diag,interface-1,-(z[interface+1]-z[interface])*e1/(e1+e2)/norm - 0.5*(z[interface]-z[interface-1])/norm);
    gsl_vector_set(up,interface-1, 0.5*(z[interface]-z[interface-1])/norm);//    +0* pow(z[interface+1]-z[interface-1],-2.0)*(1.0 - lB2/lB1));
    gsl_vector_set(low,interface-2, (z[interface+1]-z[interface])*e1/(e1+e2)/norm);//- 0*pow(z[interface+1]-z[interface-1],-2.0)*(1.0 - lB2/lB1));
*/  // boundary condition at the system boundaries (zero electric field)
  gsl_vector_set(diag,0,(z[0]-z[1] - (z[2]-z[1])*(1.0-exp(-kappa1*(z[1]-z[0])))/(z[2]-z[1])/(z[1]-z[0])/(z[2]-z[0])*2.0);
  gsl_vector_set(diag,Nz-3,(z[Nz-2]-z[Nz-1] - (z[Nz-2]-z[Nz-3])*(exp(-kappa2*(z[Nz-1]-z[Nz-2]))-1.0))/(z[Nz-1]-z[Nz-2])/(z[Nz-2]-z[Nz-3])/(z[Nz-1]-z[Nz-3])*2.0);
//  gsl_vector_set(diag,0,(z[0]-z[1])/(z[2]-z[1])/(z[1]-z[0])/(z[2]-z[0])*2.0);
//  gsl_vector_set(diag,Nz-3,(z[Nz-2]-z[Nz-1])/(z[Nz-1]-z[Nz-2])/(z[Nz-2]-z[Nz-3])/(z[Nz-1]-z[Nz-3])*2.0);


  gsl_linalg_solve_tridiag (diag,up,low,Q,phivec);

  for (j=0;j<Nz-2;j++) { newphi[j+1] = gsl_vector_get(phivec,j) - gsl_vector_get(phivec,0); }

  newphi[0] = newphi[1]*exp(-kappa1*(z[1]-z[0])); newphi[Nz-1] = (newphi[Nz-2]-phiD)*exp(-kappa2*(z[Nz-1]-z[Nz-2]))+phiD;
 // newphi[0] = newphi[1]; newphi[Nz-1] = newphi[Nz-2];

//  for (j=0;j<Nz;j++) { phi[j] -= alpha*(phi[j] - newphi[j]); }
  for (j=0;j<Nz;j++) { phi[j] -= (phi[j] - newphi[j]); }

  free(newphi);
  gsl_vector_free(Q);gsl_vector_free(diag);gsl_vector_free(up);gsl_vector_free(low);gsl_vector_free(phivec);

}

void CoulombFluid::CalculateExpensivePotential() {


  double A1,A2,B1,B2,g,sum;
  double * Q;

  Q = (double*) calloc (Nz,sizeof(double));

  for (j=0;j<Nz;j++) { Q[j] = qpv[j]*cp[j] - qmv[j]*cm[j];}

  A1 = 2*pi*lB1;  B1 = 2*lB2/(lB1+lB2);
  A2 = 2*pi*lB2;  B2 = 2*lB1/(lB1+lB2);
  g = 0.5*(B1-B2);

  for (j=1;j<Nz-1;j++) {
    sum = 0;
    for (k=1;k<Nz-1;k++) {
      if (z[k] < 0 and z[j] < 0)    sum += A1*(fabs(z[k]-z[j]) + fabs(z[k]+z[j]+epsilon)* g )*  Q[k] *dz[k];
      if (z[k] >= 0 and z[j] >= 0)  sum += A2*(fabs(z[k]-z[j]) - fabs(z[k]+z[j]+epsilon)* g )*  Q[k] *dz[k];
      if (z[k] >= 0 and z[j] < 0)   sum += A1*fabs(z[j]-z[k])*B1* Q[k] *dz[k];
      if (z[k] < 0 and z[j] >= 0)   sum += A2*fabs(z[j]-z[k])*B2* Q[k] *dz[k];
    }
    phi[j] = -sum;
//    if (j>1) phi[j] -= phi[1];
  }
  for (j=1;j<Nz-1;j++) {
    if (j!= Nz/2) phi[j] -= phi[Nz/2];
  } phi[Nz/2]=0;
  phi[0] = phi[1]; phi[Nz-1] = phi[Nz-2];

  free(Q);

}

void CoulombFluid::CalculateExpensivePotential2Walls() {


  double sum, A1;
  double * Q, * newphi, alpha2;

  Q = (double*) calloc (Nz,sizeof(double));
  newphi = (double*) calloc (Nz,sizeof(double));
```

```
  alpha2 = 1.0;
  //alpha2 = alpha;

  for (j=0;j<Nz;j++) { Q[j] = qp*cp[j] - qm*cm[j];}

  A1 = 2*pi*lB1;

  sum = 0;
  for (k=0;k<Nz;k++) {
    sum += A1*(fabs(z[k]-z[Nz/2]))*  Q[k] *dz[k];
  }
  newphi[Nz/2] = -sum;
  for (j=1;j<Nz-1;j++) {
    sum = 0;
    for (k=0;k<Nz;k++) {
      sum += A1*(fabs(z[k]-z[j]))*  Q[k] *dz[k];
    }
    newphi[j] = -sum;
    //if (j>1) newphi[j] -= newphi[1];

    //if (j!=Nz/2) newphi[j] -= newphi[Nz/2];
    phi[j] -= alpha2*(phi[j]-newphi[j]);
    //phi[j] = 0; //ALERT ALERT ALERT
  } //newphi[Nz/2] = 0; phi[Nz/2] = 0;
  phi[0] = phi[1] + 2*pi*lB1*Q[0]*pow(dz[0],2.0);  phi[Nz-1] = phi[Nz-2] + 2*pi*lB1*Q[Nz-1]*pow(dz[Nz-1],2.0);


  /*
  // For grand canonical calculations, the finite difference method performed better

  for (j=1;j<Nz-1;j++) {
    sum = phi[j+1]*dz[j] - phi[j]*(dz[j]+dz[j+1])+phi[j-1]*dz[j+1];
    sum/= 0.5*dz[j]*dz[j+1]*(dz[j]+dz[j+1])*kappa1*kappa1*exp(-0.5*(Vextm[j] + muexm[j] - muEx1m + Vextp[j] + muexp[j] - muEx1p));
    newphi[j] = 0.5*(Vextm[j] + muexm[j] - muEx1m - Vextp[j] - muexp[j] + muEx1p) + log(sum + sqrt(1.0+sum*sum)); // ALERT the external potential has been modified
  }
  for (j=1;j<Nz-1;j++) { phi[j] -= alpha2*(phi[j]-newphi[j]); }
  //  phi[0] = phi[1] + 2*pi*lB1*Q[0]*pow(dz[0],2.0); phi[Nz-1] = phi[Nz-2] + 2*pi*lB1*Q[Nz-1]*pow(dz[Nz-1],2.0);
  //  phi[0] = phi[1];  phi[Nz-1] = phi[Nz-2];
  phi[0] = phi[1] - sigma * (z[1]-z[0]);  phi[Nz-1] = phi[Nz-2] - sigma * (z[Nz-1]-z[Nz-2]);
  */
  free(Q); free (newphi);


}

void CoulombFluid::Potentiostat() {

  // Keeps the potential difference over the slit fixed. Careful with other external potentials: they may not be preserved after this step, in the current setup.

  double Vdif, scale;

  Vdif = fabs(phi[Nz-1]-phi[0]);

  Vdif *= kB*T / e;

  //if (Vdif != 0.0) sigma_renorm = 0.1 / Vdif * sigma;
  //else { sigma_renorm = sigma; cout << "careful, potential difference is zero" << endl; }

  if (Vdif != 0.0) scale = Vtot / Vdif;
  else { scale = 1.0; cout << "careful, potential difference is zero" << endl; }

  cout << "potential multiplied by " << scale << endl;


  for (j=0;j<Nz;j++) {
    Vextp[j] *= scale;
    Vextm[j] *= scale;
  }


}

void CoulombFluid::CalculateEfieldAndPotential() {


  double sum, sum1, sum2, A1;
  double * Q, * newphi, alpha2, * E, * E1, * E2;

  Q = (double*) calloc (Nz,sizeof(double));
  E = (double*) calloc (Nz,sizeof(double));
  E1 = (double*) calloc (Nz,sizeof(double));
  E2 = (double*) calloc (Nz,sizeof(double));
  newphi = (double*) calloc (Nz,sizeof(double));

  alpha2 = 0.005;
  //alpha2 = alpha;

  for (j=0;j<Nz;j++) { Q[j] = qp*cp[j] - qm*cm[j];}

  A1 = 4*pi*lB1; E[0] = E[Nz-1] = 0; E1[0] = E1[Nz-1] = 0; E2[0] = E2[Nz-1] = 0;

  for (j=1;j<Nz-1;j++) { E1[j] = E1[j-1] + A1 * Q[j-1] * dz[j];}
  for (j=Nz-2;j>0;j--) { E2[j] = E2[j+1] - A1 * Q[j+1] * dz[j];}
  for (j=1;j<Nz-1;j++) { E[j] = (E1[j]+E2[j])/2.0;}


  for (j=1;j<Nz-1;j++) {
    sum = 0; sum1 = 0; sum2 = 0;
    for (k=0;k<j+1;k++) {       sum1 += E[k] *dz[k];    }
    for (k=Nz-1;k>j-1;k--) {    sum2 += E[k] *dz[k];    }

    // newphi[j] = -sum;
    newphi[j] = -(sum1+sum2)/2.0;
```

```
    phi[j] -= alpha2*(phi[j]-newphi[j]);
  }
  phi[0] = phi[1] = phi[Nz-1] = phi[Nz-2];


/*  // For grand canonical calculations, the finite difference method performed better

  for (j=1;j<Nz-1;j++) {
    sum = phi[j+1]*dz[j] - phi[j]*(dz[j]+dz[j+1])+phi[j-1]*dz[j+1];
    sum/= 0.5*dz[j]*dz[j+1]*(dz[j]+dz[j+1])*kappa1*kappa1*exp(-0.5*(Vextm[j] + muexm[j] - muEx1m + Vextp[j] + muexp[j] - muEx1p));
    newphi[j] = 0.5*(Vextm[j] + muexm[j] - muEx1m - Vextp[j] - muexp[j] + muEx1p) + log(sum + sqrt(1.0+sum*sum)); // ALERT the external potential has been modified
  }
  for (j=1;j<Nz-1;j++) { phi[j] -= alpha2*(phi[j]-newphi[j]); }
  //  phi[0] = phi[1] + 2*pi*lB1*Q[0]*pow(dz[0],2.0); phi[Nz-1] = phi[Nz-2] + 2*pi*lB1*Q[Nz-1]*pow(dz[Nz-1],2.0);
  //  phi[0] = phi[1];  phi[Nz-1] = phi[Nz-2];
  phi[0] = phi[1] - sigma * (z[1]-z[0]);  phi[Nz-1] = phi[Nz-2] - sigma * (z[Nz-1]-z[Nz-2]);
  */
  free(Q); free(E); free(E1); free(E2); free (newphi);


}

void CoulombFluid::CalculatePotentialWithTraceSalt() {


  double sum, A1, rhoTr;
  double * Q, * newphi, alpha2, *DDphi;

  Q = (double*) calloc (Nz,sizeof(double));
  newphi = (double*) calloc (Nz,sizeof(double));
  DDphi = (double*) calloc (Nz,sizeof(double));

  alpha2 = 1.0;//0.005;
  rhoTr= 10e-4;
  //alpha2 = alpha;

  for (j=0;j<Nz;j++) { Q[j] = qp*cp[j] - qm*cm[j] + rhoTr*cs1*exp(-2*phi[j]) + QDNA[j];}

  A1 = 4.0*pi*lB1;

  for (j=1;j<Nz-1;j++) { DDphi[j] = phi[j+1] + phi[j-1] - 2*phi[j]; DDphi[j] /= dz[j]*dz[j];}

  for (j=1;j<Nz-1;j++) {

    sum = DDphi[j] / A1 + Q[j];
    sum /= (2.0*rhoTr);

    newphi[j] = log(sum + sqrt(1.0+sum*sum));
    phi[j] -= alpha2*(phi[j]-newphi[j]);
  }
  phi[0] = phi[1]; phi[Nz-1] = phi[Nz-2];


/*  // For grand canonical calculations, the finite difference method performed better

  for (j=1;j<Nz-1;j++) {
    sum = phi[j+1]*dz[j] - phi[j]*(dz[j]+dz[j+1])+phi[j-1]*dz[j+1];
    sum/= 0.5*dz[j]*dz[j+1]*(dz[j]+dz[j+1])*kappa1*kappa1*exp(-0.5*(Vextm[j] + muexm[j] - muEx1m + Vextp[j] + muexp[j] - muEx1p));
    newphi[j] = 0.5*(Vextm[j] + muexm[j] - muEx1m - Vextp[j] - muexp[j] + muEx1p) + log(sum + sqrt(1.0+sum*sum)); // ALERT the external potential has been modified
  }
  for (j=1;j<Nz-1;j++) { phi[j] -= alpha2*(phi[j]-newphi[j]); }
  //  phi[0] = phi[1] + 2*pi*lB1*Q[0]*pow(dz[0],2.0); phi[Nz-1] = phi[Nz-2] + 2*pi*lB1*Q[Nz-1]*pow(dz[Nz-1],2.0);
  //  phi[0] = phi[1];  phi[Nz-1] = phi[Nz-2];
  phi[0] = phi[1] - sigma * (z[1]-z[0]);  phi[Nz-1] = phi[Nz-2] - sigma * (z[Nz-1]-z[Nz-2]);
  */
  free(Q); free (newphi); free(DDphi);


}

void CoulombFluid::CalculateAnalyticalProfiles() {

  double C1, C2, n, p, D;

  n = sqrt(e1/e2); p = sqrt(cs1/cs2); D = sqrt(n*n*p*p + 2*n*p*cosh(phiD/2.0)+1.0);

  C1 = n*p + cosh(phiD/2.0) - D;     C1 /= sinh(phiD/2.0);
  C2 = -1.0 - n*p*cosh(phiD/2.0) + D;    C2 /= n*p*sinh(phiD/2.0);

  for (j=0;j<interface;j++) {
    phiAN[j] = 4.0*atanh(C1*exp(kappa1*(z[j]+epsilon)));
    cpAN[j] = cs1*exp(-phiAN[j]);
    cmAN[j] = qp/qm*cs1*exp( phiAN[j]);
  }
  for (j=interface;j<Nz;j++) {
    phiAN[j] = 4.0*atanh(C2*exp(-kappa2*(z[j]+epsilon)))+phiD;
    cpAN[j] = cs2*exp(-phiAN[j] + phiD);
    cmAN[j] = qp/qm*cs2*exp( phiAN[j] - phiD);
  }

  for (j=0;j<Nz;j++) {  phi[j] = phiAN[j];    cp[j] = cpAN[j];    cm[j] = cmAN[j];  }

}

void CoulombFluid::CalculateFreeEnergy() {

  gsl_dht * t = gsl_dht_new(Nr, 0.0, r[Nr-1]);
  gsl_matrix * m = gsl_matrix_alloc (2*Nz, 2*Nz);
  gsl_permutation * p = gsl_permutation_alloc (2*Nz);
  int sign[1];
  double det, Fp, Fm;

  Pzz[nf] = cp[0] + cm[0] - (Vextp[1]-Vextp[0])/dz[0]*cp[0] - (Vextm[1]-Vextm[0])/dz[0]*cm[0];
  for (j=1;j<Nz/2;j++) {
    Fp = - (Vextp[j+1]-Vextp[j-1])/(2.0*dz[j]);
```

```
  Fm = - (Vextm[j+1]-Vextm[j-1])/(2.0*dz[j]);
  Pzz[nf] += Fp*n[j] + Fm*n[j+Nz];
}


for (j=0;j<2*Nz;j++) { for (k=0;k<2*Nz;k++) {
  Zcorr[j][k] = mu[j][k] = 0.0;
}}

for (j=0;j<2*Nz;j++) { for (k=0;k<2*Nz;k++) {
  for (i=1;i<Nr;i++) {
    mu[j][k] -= 2*pi*r[i]*(r[i] - r[i-1]) * (0.5*pow(h[i][j][k]-Qs[i][j][k],2.0) + h[i][j][k]-Qs[i][j][k]);
    mu[j][k] += 2*pi*r[i]*(r[i] - r[i-1]) * ( (h[i][j][k]-Qs[i][j][k]+1.0)*log(h[i][j][k]-Qs[i][j][k]+1.0+1e-30) + (h[i][j][k]-Qs[i][j][k]+1.0)*u[i][j][k] );
  }
}}

for (j=0;j<Nz;j++) { for (k=0;k<Nz;k++) {
  for (i=1;i<Nr;i++) {
    Zcorr[j][k] += 2*pi*r[i]*(r[i] - r[i-1]) * ( (h[i][j][k]-Qs[i][j][k]+1.0)     *n[j]*n[k]  + (h[i][j+Nz][k+Nz]-Qs[i][j+Nz][k+Nz]+1.0)  *n[j+Nz]*n[k+Nz] );
    Zcorr[j][k] -= 2*pi*r[i]*(r[i] - r[i-1]) * ( (h[i][j][k+Nz]-Qs[i][j][k+Nz]+1.0)   *n[j]*n[k+Nz]   + (h[i][j+Nz][k]-Qs[i][j+Nz][k]+1.0)        *n[j+Nz]*n[k] );
  }
}}

for (j=0;j<Nz;j++) {
  if (z[j]-z[0] < Vlength) NP[nf] += n[j];
  if (z[j]-z[0] < Vlength) NM[nf] += n[j+Nz];
}

for (j=0;j<2*Nz;j++) {
  for (k=0;k<2*Nz;k++) {
    Fexc[nf] += 0.5*n[j]*n[k]*mu[j][k];
  }
}

for (j=0;j<Nz;j++) {
  for (k=0;k<Nz;k++) {
    if (z[j]-z[0]    < Vlength and z[k]-z[0]    < Vlength) Z1Z1[nf] += Zcorr[j][k];
    if (z[j]-z[0]    < Vlength and z[Nz-1]-z[k] < Vlength) Z1Z2[nf] += Zcorr[j][k];
    if (z[Nz-1]-z[j] < Vlength and z[Nz-1]-z[k] < Vlength) Z2Z2[nf] += Zcorr[j][k];
  }
}

for (j=0;j<2*Nz;j++) { for (k=0;k<2*Nz;k++) {
  for (i=0;i<Nr;i++) { dc[i] = h[i][j][k]; }
  gsl_dht_apply(t, dc, dcHT);
  for (i=0;i<Nr;i++) { hFT[i][j][k] = dcHT[i]*2*pi - QsHT[i][j][k];}
}}

for (i=0;i<Nr-1;i++) {
  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
    gsl_matrix_set (m, j, k, delta[j][k] + hFT[i][j][k]*n[k]);
  }}
  gsl_linalg_LU_decomp(m,p,sign);
    det = gsl_linalg_LU_lndet (m);
    for (j=0;j<2*Nz;j++) det -= n[j]*hFT[i][j][j];
    Fexc[nf] -= 0.5*pow(2*pi,-2.0)*(kFT[i+1]-kFT[i])*2*pi*kFT[i]*det; /* factor (1/2*pi)^2 to rescale k appropriately, to dimension [1/nm]; HT defined such that f(r)=1/(2pi)^2*Int_vector{k} f(k) */
  }

  for (j=0;j<Nz;j++) {
   Fid[nf] += (cp[j]*(log(cp[j]/cs1)-1.0) + cm[j]*(log(cm[j]/cs1)-1.0))*dz[j];
   Fext[nf]+= (cp[j]*(Vextp[j]+phi[j]) + cm[j]*(Vextm[j]-phi[j]))*dz[j];
  }

  Hf[nf] = H/kappa1; cout << "TEST: H/kappa1 = " << H/kappa1 << " and H[hf] = " << Hf[nf] << endl;

  FexcDH[nf] = sqrt(8*pi*lB1*qm*cm[Nz/2]);  FexcDH[nf] = -pow(FexcDH[nf],3.0)/(12*pi);

  gsl_matrix_free (m); gsl_permutation_free (p); gsl_dht_free (t);

}

void CoulombFluid::CalculateCorrelationLength() {

  double *Lambda,*Inverse,*Inverse2,range[2];
  int l[2],l1,l2;

  Lambda = (double*) calloc (Nz,sizeof(double));
  Inverse = (double*) calloc (Nz,sizeof(double));
  Inverse2 = (double*) calloc (Nz,sizeof(double));
  range[0] = 2*ap + 2.0/kappa1;
  range[1] = 2*ap + 4.0/kappa1;

  i=0;
  while(i<Nr-1 and r[i] < range[0])i++;
  l[0] = i; cout << "l[0] = " << l[0] << " and r[l[0]] = " << r[l[0]] << " nm." << endl;
  l1=i;
  while(i<Nr-1 and r[i] < range[1])i++;
  l[1] = i;
  l2=l1+5;

  for (j=0;j<Nz;j++) {
    //Lambda[j] = log(h[l[0]][j][j+Nz]*r[l[0]]) - log(h[l[1]][j][j+Nz]*r[l[1]]);    Lambda[j] /= r[l[1]]-r[l[0]];   Lambda[j] = 1.0/Lambda[j];
    Lambda[j] = log(h[l1][j][j+Nz]*r[l1]) - log(h[l2][j][j+Nz]*r[l2]);      Lambda[j] /= r[l2]-r[l1];   Lambda[j] = 1.0/Lambda[j];
    Lambda[j] *= kappa1;
  }

  for (j=0;j<Nz;j++) {
    i=1;
    while (i<Nr-1 and h[i][j][j+Nz]<0) i++;
    while (i<Nr-1 and h[i][j][j+Nz]>=0) i++;
    if(i<Nr) Inverse[j] = r[i];
    else Inverse[j] = r[Nr-1];
  }

  for (j=0;j<Nz;j++) {
```

```
    if(j>Nz/2) { Inverse2[j]=Inverse2[Nz-j]; continue; }
    k=j;
    while (k<Nz-1 and h[0][j][k+Nz]<0) k++;
    while (k<Nz-1 and h[0][j][k+Nz]>=0) k++;
    if(k<Nz) Inverse2[j] = z[k]-z[j];
    else Inverse2[j] = z[Nz-1]-z[j];
  }

  ofstream datafile;
  char filename[70];

  strcpy(filename,"Data/"); strcat(filename, project); strcat(filename,"_CorrelationLength.dat");

  datafile.open(filename,ios::trunc);
  datafile << "#" << setw(29) << "1) z (nm)" << setw(30) << "2) Lambda parallel*" << setw(30) << "3) First zero parallel (nm)";
  datafile << setw(30) << "4) Wavelength/Lambda  " << " *(divided by the Debye length of a homogeneous system)" << endl;
  for (j=0;j<Nz;j++) {
    datafile << setw(30) << z[j] << setw(30) << Lambda[j] << setw(30) << Inverse[j] << setw(30) << (Inverse[j]-(ap+am))/Lambda[j];
    datafile << setw(30) << Inverse2[j] << setw(30) << (Inverse2[j]-(ap+am))/Lambda[j] << endl;
  }
  datafile.close();

  free(Lambda); free(Inverse); free(Inverse2);
}

void CoulombFluid::SymmetrizeFunctions(int NzMem) {

  /*
  h.assign(0.0,Nr,2*Nz,2*Nz);
  c.assign(0.0,Nr,2*Nz,2*Nz);
  y.assign(0.0,Nr,2*Nz,2*Nz);
  u.assign(0.0,Nr,2*Nz,2*Nz);
  uHT.assign(0.0,Nr,2*Nz,2*Nz);
  uReg.assign(0.0,Nr,2*Nz,2*Nz);
  uRegHT.assign(0.0,Nr,2*Nz,2*Nz);
  newh.assign(0.0,Nr,2*Nz,2*Nz);
  hInit.assign(0.0,Nr,2*Nz,2*Nz);
  hFin.assign(0.0,Nr,2*Nz,2*Nz);
  newg.assign(0.0,Nr,2*Nz,2*Nz);
  cInit.assign(0.0,Nr,2*Nz,2*Nz);
  cFin.assign(0.0,Nr,2*Nz,2*Nz);
  newc.assign(0.0,Nr,2*Nz,2*Nz);
  Qs.assign(0.0,Nr,2*Nz,2*Nz);
  QsHT.assign(0.0,Nr,2*Nz,2*Nz);
  hFT.assign(0.0,Nr,2*Nz,2*Nz);

  delta.assign(0.0,2*Nz,2*Nz);
  htest.assign(0.0,Nr,2*Nz);
  mu.assign(0.0,2*Nz,2*Nz);
  newMu.assign(0.0,2*Nz,2*Nz);
  Zcorr.assign(0.0,2*Nz,2*Nz);
  for (i=0;i<2*Nz;i++) delta[i][i] = 1.0;
  n.assign(0.0,2*Nz);
  nOld.assign(0.0,2*Nz);
  z.assign(0.0,Nz);
  dz.assign(0.0,Nz);
  r.assign(0.0,Nr);
  kFT.assign(0.0,Nr);
  */

  for (j=0;j<Nz;j++) {  for (k=0;k<Nz;k++) {  for (i=0;i<Nr;i++) {
      h[i][Nz-1-j][k] = h[i][j][Nz-1-k];
      h[i][2*Nz-1-j][k] = h[i][j][2*Nz-1-k];
      h[i][2*Nz-1-j][k+Nz] = h[i][j+Nz][2*Nz-1-k];
      h[i][Nz-1-j][Nz-1-k] = h[i][j][k];
      h[i][2*Nz-1-j][2*Nz-1-k] = h[i][Nz+j][Nz+k];
      c[i][Nz-1-j][k] = c[i][j][Nz-1-k];
      c[i][Nz-1-j][Nz-1-k] = c[i][j][k];
  }}
    n[Nz-1-j]=n[j]; n[2*Nz-1-j]=n[Nz+j];
    cp[Nz-1-j]=cp[j]; cm[Nz-1-j]=cm[j];
  }

  for (j=0;j<2*Nz;j++) {  for (k=0;k<2*Nz;k++) {  for (i=0;i<Nr;i++) {
      h[i][j][k] = c[i][j][k] = Qs[i][j][k] = u[i][j][k] = uHT[i][j][k] = uReg[i][j][k] = uRegHT[i][j][k] = 0.0;
  }}}
  for (j=0;j<Nz;j++) {
//    n[Nz-1-j]=n[j]; n[2*Nz-1-j]=n[Nz+j];
    cp[j]=cs1*exp(-Vextp[j]); cm[j]=cs1*qp/qm*exp(-Vextm[j]);
    n[j]=cp[j]*dz[j]; n[Nz+j]=cm[j]*dz[j];
//    cp[Nz-1-j]=cp[j]; cm[Nz-1-j]=cm[j];
  }

}

void CoulombFluid::Iterate() {


  time_t start,end;
  int Time[3];
  time (&start);
  bool append=true;

  InformationFile(AskQuestion);

  cout << "\n\n *** Begin of iteration *** \n\n" << endl;

  correlations = false;

//  SetGridExcluded();
  SetGrid2Walls();
  SetPairPotential();
//  CalculateAnalyticalProfiles();
  CalculateDensity(3e4);
```

```
  RecordProfiles(!append);

//  RecordSnapShot();

  correlations = true;
  SetHankelTransformPairPotential();
//  EvaluateReservoirMuEx(1);//  EvaluateReservoirMuEx(2);

  while (!convergence) {

    OZ_HNC_Alternative();
    CalculateDensity(3e4);
    RecordProfiles(!append);

//    RecordSnapShot();
  }

  RecordProfiles(!append);
  RecordCorrelationFunctions();
  time (&end);

  Time[0] = int(difftime (end,start)/3600.0);   Time[1] = int(int(difftime (end,start))%3600/60.0);   Time[2] = int(difftime (end,start))%60;
  RecordTime (Time);
  cout << "\n\n *** End of iteration *** \n\n" << endl;

}

void CoulombFluid::Iterate2Walls() {

  time_t start,end;
  int Time[3];
  time (&start);
  bool append=true;

  InformationFile(AskQuestion);

  cout << "\n\n *** Begin of iteration *** \n\n" << endl;

  correlations = false;

  depth = 0*1.5;

  SetGrid2Walls();
  SetPairPotential2Walls();
  CalculateDensity(3e4);
  RecordProfiles(!append);

  //  RecordSnapShot();

  correlations = true;
  //  SetHankelTransformPairPotential2Walls();
  EvaluateReservoirMuEx(1);//  EvaluateReservoirMuEx(2);

  while (!convergence) {

    OZ_HNC_Alternative();
    CalculateDensity(3e4);
    RecordProfiles(!append);

  }

  RecordCorrelationFunctions();
  RecordTestfunctions();
  time (&end);

  Time[0] = int(difftime (end,start)/3600.0);   Time[1] = int(int(difftime (end,start))%3600/60.0);   Time[2] = int(difftime (end,start))%60;
  RecordTime (Time);
  cout << "\n\n *** End of iteration *** \n\n" << endl;

}

void CoulombFluid::Iterate2DielectricWalls() {

  time_t start,end;
  int Time[3], write=0;
  time (&start);
  bool append=true;
  double precision = 1e-3;

  InformationFile(AskQuestion);

  cout << "\n\n *** Begin of iteration *** \n\n" << endl;

  correlations = false;

  GridCutoff = 1.7; Htot = 3.0;
  depth = 0.0; Lw = 0.0; H=(Htot-GridCutoff*ap)*kappa1;
  //depth = 0.0; Lw = 0.0; H=3.0;

  SetGrid2Walls();
  SetPairPotential2DielectricWalls();

  //OZ_AHNC_AdaptiveFinder(precision); GnuplotCorrelationFunctions();
  CalculateDensity(7e4);
  RecordProfiles(!append);

    //RecordSnapShot();

  correlations = true;
  //  SetHankelTransformPairPotential2Walls();
  //EvaluateReservoirMuEx(1);//  EvaluateReservoirMuEx(2);

  while (!convergence) {

    OZ_AHNC_AdaptiveFinder(precision); GnuplotCorrelationFunctions();
```

```
    //CalculateCorrelationLength();
    CalculateDensity(7e4);

    RecordProfiles(!append);
    //if (convergence and depth < 4.0) { depth += 0.1; cout << "\ndepth = " << depth << endl; convergence=false; SetPairPotential2DielectricWalls(); CalculateDensity(3e4); }
    //if (convergence and depth < 1.0) { depth += 0.05; CalculateDensity(3e4); }
   //convergence = true;
    if(write%10 == 0) { RecordCorrelationFunctions();}
    write++;

    cout << "\n\nNext iteration of \"" << project << "\"\n" << endl;

  }

  RecordCorrelationFunctions();
  RecordTestfunctions();
  time (&end);

  Time[0] = int(difftime (end,start)/3600.0);   Time[1] = int(int(difftime (end,start))%3600/60.0);   Time[2] = int(difftime (end,start))%60;
  RecordTime (Time);
  cout << "\n\n *** End of iteration *** \n\n" << endl;

  wait_for_key(); wait_for_key();

}

void CoulombFluid::Iterate2Walls1Dielectric() {

  time_t start,end;
  int Time[3], t0=1,tm;
  time (&start);
  bool append=true;
  double precision = 1e-3;


  depth = 0.0; Lw = 0.0; H=(3.0-1.7*ap)*kappa1;

  temp = t0;    ResetValues();

  InformationFile(AskQuestion);

  cout << "\n\n *** Begin of iteration *** \n\n" << endl;

  correlations = false;

//  SetGridExcluded();
//  SetGrid2Walls1Dielectric();
  SetGrid2Walls();
  //SetPairPotential2Walls1Dielectric();
  SetPairPotential2DielectricWalls();
//  CalculateAnalyticalProfiles();
  CalculateDensity(3e4);
  RecordProfiles(!append);
//  RecordSnapShot();

  correlations = true;
  //EvaluateReservoirMuEx(1);
//  EvaluateReservoirMuEx(2);

  for (tm=0;tm<t0;tm++) {

    convergence = false; evaluations = 0;
    temp = t0/(tm+1.0);
    if (temp > 1.0) cout << "\n\n|| Temperature raised by a factor of " << temp << " ||" << endl;
    if (temp ==1.0) { cout << "Room temperature" << endl; precision *= 1e-1; }
    if (tm>0) cout << "\nCooled down by a factor of " << (tm+2.0)/(tm+1.0) << endl;
    cout << "tm = " << tm << endl;

    ResetValues();
    SetGrid2Walls();
    //SetPairPotential2Walls1Dielectric();
    SetPairPotential2DielectricWalls();
    //SetHankelTransformPairPotential();

    InformationFile(!AskQuestion);

    while (!convergence) {

      //EvaluateReservoirMuEx(1);
      OZ_AHNC_AdaptiveFinder(precision);
      CalculateDensity(3e4); if (Singularities) break;
      RecordProfiles(!append);

    }
    if (Singularities) break;
  }

  if (!Singularities) {
    RecordProfiles(!append);
    RecordCorrelationFunctions();
  }
  time (&end);

  Time[0] = int(difftime (end,start)/3600.0);   Time[1] = int(int(difftime (end,start))%3600/60.0);   Time[2] = int(difftime (end,start))%60;
  RecordTime (Time);
  if (Singularities) cout << "\n\n *** End of iteration because of singularities *** \n\n" << endl;
  else cout << "\n\n *** End of iteration *** \n\n" << endl;
}

void CoulombFluid::Iterate2WallSession() {

  time_t start,end;
  int Time[3], NzMem, NzMem2, NfSteps=30;
  time (&start);
```

```
  bool append=true;
  double precision = 1e-3;

  Lw=2.0; /* layer width in nm */
  H0 = H = 2.0;
  NzMem = Nz;

  InformationFile(AskQuestion);

  cout << "\n\n *** Begin of iteration *** \n\n" << endl;

  correlations = false;

  depth = 1.3;
  SetGrid2WallSession(NzMem);
  SetPairPotential2Walls();
  CalculateDensity(3e4);
  RecordProfiles(!append);

  correlations = true;
  EvaluateReservoirMuEx(1);

  while (nf < Nf) {

    evaluations = 0; convergence = false;

    H = H0 - ((nf%NfSteps)*(H0-2.0*kappa1)/(NfSteps*1.0-1.0));
    NzMem2=Nz;
    if (nf%NfSteps == 0 and nf !=0) {
      Nz = NzMem;
      depth += 0.2;
      correlations = false;

      SetGrid2WallSession(NzMem);
      SetPairPotential2Walls();
      CalculateDensity(3e4);
//      RecordProfiles(append);

      correlations = true;
      EvaluateReservoirMuEx(1);
    }
    cout << "\nH = " << H/kappa1 << " nm, and nf = " << nf << endl;

    SetGrid2WallSession(NzMem);
    SetPairPotential2Walls();
//    NzMem2 = Nz; //for (j=Nz;j<NzMem;j++) { z[j] = z[Nz-1]; cp[j] = cp[Nz-1]; cm[j] = cm[Nz-1]; phi[j] = phi[Nz-1]; }
//    Nz=NzMem;
    SymmetrizeFunctions(NzMem2); /* NB: it is crucial for convergence to put the Qs-matrix to zero before the new iteration. Is done in this routine. */
//    Nz = NzMem2;

//    correlations = false;      CalculateDensity(3e4);      correlations = true;
    CalculateDensity(3e4);

    while (!convergence) {

//      OZ_HNC_Alternative();
      OZ_AHNC_AdaptiveFinder(precision);
      CalculateDensity(3e4);
      if (nf==0) { RecordFreeEnergy(!append); RecordProfiles(!append); }
    }

    if (nf==0) {
      NzMem2 = Nz; for (j=Nz;j<NzMem;j++) { z[j] = z[Nz-1]; n[j] = n[Nz-1]; n[j+Nz] = n[2*Nz-1]; cp[j] = cp[Nz-1]; cm[j] = cm[Nz-1]; phi[j] = phi[Nz-1]; } Nz=NzMem;
      RecordProfiles(!append);
      Nz = NzMem2;
    } else {
      NzMem2 = Nz; for (j=Nz;j<NzMem;j++) { z[j] = z[Nz-1]; n[j] = n[Nz-1]; n[j+Nz] = n[2*Nz-1]; cp[j] = cp[Nz-1]; cm[j] = cm[Nz-1]; phi[j] = phi[Nz-1]; } Nz=NzMem;
      RecordProfiles(append);
      Nz = NzMem2;
    }
    RecordCorrelationFunctions();
    CalculateFreeEnergy();
    RecordFreeEnergy(append);

  } /*end of 'while' loop*/

  time (&end);

  Time[0] = int(difftime (end,start)/3600.0);   Time[1] = int(int(difftime (end,start))%3600/60.0);   Time[2] = int(difftime (end,start))%60;
  RecordTime (Time);
  cout << "\n\n *** End of iteration *** \n\n" << endl;

}

void CoulombFluid::Iterate2DielectricWallSession() {

  time_t start,end;
  int Time[3], NzMem, NzMem2, NfSteps=Nf;
  time (&start);
  bool append=true;
  double LDNA, precision = 1e-4,eInit;

  //eInit=70.0; e2=e3=eInit;
  //eInit=ew; e2=e3=eInit;

  //Lw=0.9; /* layer width in nm */
  Lw=0.0;
  LDNA = 1.0; // length of DNA chain in nm

  alpha = 0.001;
  sigmaDNA = -0.01;
  GridCutoff = 2.0;
  Htot = 3.0+(2.0*ap+1.0*LDNA)*kappa1; H0 = H = (Htot-GridCutoff*ap)*kappa1;
  NzMem = Nz;
```

```
  InformationFile(AskQuestion);

  cout << "\n\n *** Begin of iteration *** \n\n" << endl;

  correlations = false;

  depth = 0.0;
  //SetGrid2WallSession(NzMem);
  SetGrid2Walls(); SetQDNA(LDNA);
  //SetGrid2Walls1Dielectric();
  //SetPairPotential2DielectricWalls();
  SetRegularizedPairPotential2DielectricWalls();
  CalculateDensity(5e4);
  RecordProfiles(!append);

  correlations = true;
  //EvaluateReservoirMuEx(1);

  //e2 /= ew/eInit;
  //e2 *= ew/eInit; e3 = ew*ew/eInit;
  EvaluateReservoirMuEx(1);
  while (nf < Nf) {

    evaluations = 0; convergence = false;

    //H = H0 - ((nf%NfSteps)*(H0-2*ap*kappa1)/(NfSteps*1.0-1.0));
    H = H0 - ((nf%NfSteps)*(H0-LDNA)/(NfSteps*1.0));
    //e2 *= 0.9;    e3=e2;
    NzMem2=Nz;
    if (nf%NfSteps == 0 and nf !=0) {
      Nz = NzMem;
      e2 *= ew/eInit; e3=e2;
      if (e2>ew*ew/eInit) { e2 = eInit; e3 = ew*ew/eInit; }
      //depth += 0.2;
      correlations = false;

      //SetGrid2WallSession(NzMem);
      SetGrid2Walls(); SetQDNA(LDNA);
      //SetGrid2Walls1Dielectric();
      //SetPairPotential2DielectricWalls();
      SetRegularizedPairPotential2DielectricWalls();
      //SymmetrizeFunctions(NzMem); /* NB: it is crucial for convergence to put the Qs-matrix to zero before the new iteration. Is done in this routine. */
      CalculateDensity(5e4);
//      RecordProfiles(append);

      correlations = true;
    }

    //correlations = false;//ALERT

    cout << "\nH = " << H/kappa1 << " nm, and nf = " << nf << endl;
    //SetGrid2WallSession(NzMem);
    SetGrid2Walls(); SetQDNA(LDNA);
    //SetGrid2Walls1Dielectric();
    //SetPairPotential2DielectricWalls();
    SetRegularizedPairPotential2DielectricWalls();
      //    NzMem2 = Nz; //for (j=Nz;j<NzMem;j++) { z[j] = z[Nz-1]; cp[j] = cp[Nz-1]; cm[j] = cm[Nz-1]; phi[j] = phi[Nz-1]; }
//    Nz=NzMem;
    //SymmetrizeFunctions(NzMem); /* NB: it is crucial for convergence to put the Qs-matrix to zero before the new iteration. Is done in this routine. */
//    Nz = NzMem2;

//    correlations = false;      CalculateDensity(3e4);      correlations = true;
    CalculateDensity(5e4);

    //convergence = true;
    while (!convergence) {

//      OZ_HNC_Alternative();
      OZ_AHNC_AdaptiveFinder(precision);
      CalculateDensity(5e4);
      if (nf==0) { RecordFreeEnergy(!append); RecordProfiles(!append); }
      RecordTemporaryProfiles();
    }

    if (nf==0) {
      NzMem2 = Nz; for (j=Nz;j<NzMem;j++) { z[j] = z[Nz-1]; n[j] = n[Nz-1]; n[j+Nz] = n[2*Nz-1]; cp[j] = cp[Nz-1]; cm[j] = cm[Nz-1]; phi[j] = phi[Nz-1]; } Nz=NzMem;
      RecordProfiles(!append);
      Nz = NzMem2;
    } else {
      NzMem2 = Nz; for (j=Nz;j<NzMem;j++) { z[j] = z[Nz-1]; n[j] = n[Nz-1]; n[j+Nz] = n[2*Nz-1]; cp[j] = cp[Nz-1]; cm[j] = cm[Nz-1]; phi[j] = phi[Nz-1]; } Nz=NzMem;
      RecordProfiles(append);
      Nz = NzMem2;
    }
    RecordCorrelationFunctions();
    CalculateFreeEnergy();
    RecordFreeEnergy(append);

    nf++;

  } /*end of 'while' loop*/

  time (&end);

  Time[0] = int(difftime (end,start)/3600.0);   Time[1] = int(int(difftime (end,start))%3600/60.0);   Time[2] = int(difftime (end,start))%60;
  RecordTime (Time);
  cout << "\n\n *** End of iteration *** \n\n" << endl;

}

void CoulombFluid::RecordProfiles(bool append) {

  ofstream datafile;
  char filename[90];

  strcpy(filename,"Data/"); strcat(filename, project); strcat(filename,"_Profiles.dat");
```

```
  if (!append) {
    //cout << "IS THERE SOMETHING HAPPENING HERE?" << endl;
    datafile.open(filename,ios::trunc);
    datafile << "#" << setw(13) << "1) z (nm)" << setw(14) << "2) np (nm^-2)" << setw(14) << "3) nm (nm^-2)" << setw(14) << "4) cpMF (M)" << setw(14) << "5) cmMF (M)" << setw(14) << "6) cp (M)" << setw(14) << "7) cm (M)" << setw(14) << "8) phiAN (mV)" << setw(14) << "9) phi (mV)" <<
    for (j=0;j<Nz;j++) {
      datafile << setw(14) << z[j] << setw(14) << n[j]/NA << setw(14) << n[j+Nz]/NA << setw(14) << cpMF[j]/NA << setw(14) << cmMF[j]/NA;
      datafile << setw(14) << cp[j]/NA << setw(14) << cm[j]/NA << setw(14) << phiAN[j]*kB*T/e*1000 << setw(14) << phi[j]*kB*T/e*1000 << setw(14) << muexp[j];
      datafile << setw(14) << muexm[j] << setw(14) << Vextp[j] << setw(14) << Vextm[j] << setw(14) << h[0][j][0] << setw(14) << h[0][j][Nz/2];
      datafile << setw(14) << c[0][j][0] << setw(14) << c[0][j][Nz/2];
      if(j<Nr) datafile << setw(14) << y[j][Nz/2][Nz/2] << setw(14) << u[j][Nz/2][Nz/2] << setw(14) << uReg[j][Nz/2][Nz/2] << endl;
      else datafile << setw(14) << y[Nr-1][Nz/2][Nz/2] << setw(14) << u[Nr-1][Nz/2][Nz/2] << setw(14) << uReg[Nr-1][Nz/2][Nz/2] << endl;
    }
    datafile.close();
  } else {
    datafile.open(filename,ios::app);
    for (j=0;j<Nz;j++) {
      datafile << setw(14) << z[j] << setw(14) << n[j]/NA << setw(14) << n[j+Nz]/NA << setw(14) << cpMF[j]/NA << setw(14) << cmMF[j]/NA;
      datafile << setw(14) << cp[j]/NA << setw(14) << cm[j]/NA << setw(14) << phiAN[j]*kB*T/e*1000 << setw(14) << phi[j]*kB*T/e*1000 << setw(14) << muexp[j];
      datafile << setw(14) << muexm[j] << setw(14) << Vextp[j] << setw(14) << Vextm[j] << setw(14) << h[0][j][0] << setw(14) << h[0][j][Nz/2];
      datafile << setw(14) << c[0][j][0] << setw(14) << c[0][j][Nz/2];
      if(j<Nr) datafile << setw(14) << y[j][Nz/2][Nz/2] << setw(14) << u[j][Nz/2][Nz/2] << setw(14) << uReg[j][Nz/2][Nz/2] << endl;
      else datafile << setw(14) << y[Nr-1][Nz/2][Nz/2] << setw(14) << u[Nr-1][Nz/2][Nz/2] << setw(14) << uReg[Nr-1][Nz/2][Nz/2] << endl;
    }
    datafile.close();
  }

}

void CoulombFluid::RecordTemporaryProfiles() {

  ofstream datafile;
  char filename[90];

  strcpy(filename,"Data/"); strcat(filename, project); strcat(filename,"_TemporaryProfiles.dat");

  datafile.open(filename,ios::trunc);
  datafile << "#" << setw(13) << "1) z (nm)" << setw(14) << "2) np (nm^-2)" << setw(14) << "3) nm (nm^-2)" << setw(14) << "4) cpMF (M)" << setw(14) << "5) cmMF (M)" << setw(14) << "6) cp (M)" << setw(14) << "7) cm (M)" << setw(14) << "8) phiAN (mV)" << setw(14) << "9) phi (mV)" << se
  for (j=0;j<Nz;j++) {
    datafile << setw(14) << z[j] << setw(14) << n[j]/NA << setw(14) << n[j+Nz]/NA << setw(14) << cpMF[j]/NA << setw(14) << cmMF[j]/NA;
    datafile << setw(14) << cp[j]/NA << setw(14) << cm[j]/NA << setw(14) << phiAN[j]*kB*T/e*1000 << setw(14) << phi[j]*kB*T/e*1000 << setw(14) << muexp[j];
    datafile << setw(14) << muexm[j] << setw(14) << Vextp[j] << setw(14) << Vextm[j] << setw(14) << h[0][j][0] << setw(14) << h[0][j][Nz/2];
    datafile << setw(14) << c[0][j][0] << setw(14) << c[0][j][Nz/2];
    if(j<Nr) datafile << setw(14) << y[j][Nz/2][Nz/2] << setw(14) << u[j][Nz/2][Nz/2] << setw(14) << uReg[j][Nz/2][Nz/2] << endl;
    else datafile << setw(14) << y[Nr-1][Nz/2][Nz/2] << setw(14) << u[Nr-1][Nz/2][Nz/2] << setw(14) << uReg[Nr-1][Nz/2][Nz/2] << endl;
  }
  datafile.close();

}


void CoulombFluid::RecordSnapShot() {

double *x,*y,*zpos, Area, norm1, norm2;
gsl_rng * r;
int Npsim, Nplus, Nmin, n, *type;
const gsl_rng_type * Type;
ofstream datafile;
char filename[70];

Npsim = 200;

x = (double*) calloc (4*Npsim,sizeof(double));  y = (double*) calloc (4*Npsim,sizeof(double));  zpos = (double*) calloc (4*Npsim,sizeof(double));
type = (int*) calloc (4*Npsim,sizeof(int));

norm1 = 0;  norm2 = 0;
for (j=0;j<Nz;j++) { norm1 += dz[j]; norm2 += cp[j]*dz[j]; }

Area = Npsim/(norm2);


gsl_rng_env_setup();
Type = gsl_rng_default;
r = gsl_rng_alloc (Type);

n=0;
for (j=0;j<Nz;j++) {
  for (i=0;i<cp[j]*dz[j]*Area;i++) {
    x[n] = gsl_rng_uniform_pos (r) * sqrt(Area);
    y[n] = gsl_rng_uniform_pos (r) * sqrt(Area);
    zpos[n] = gsl_rng_uniform_pos (r) * dz[j] + z[j];
    type[n] = j;
    n++;
    if (n>=2*Npsim) {cout << "Alert: number of points exceeds reserved memory." << endl; n--;}
  }
}
Nplus = n;
for (j=0;j<Nz;j++) {
  for (i=0;i<cm[j]*dz[j]*Area;i++) {
    x[n] = gsl_rng_uniform_pos (r) * sqrt(Area);
    y[n] = gsl_rng_uniform_pos (r) * sqrt(Area);
    zpos[n] = gsl_rng_uniform_pos (r) * dz[j] + z[j];
    type[n] = j;
    n++;
    if (n>=4*Npsim) {cout << "Alert: number of points exceeds reserved memory." << endl; n--;}
  }
}
Nmin = n;

printf ("Random number generator type: %s\n", gsl_rng_name (r));
printf ("seed = %lu\n", gsl_rng_default_seed);
printf ("first value = %f\n", gsl_rng_uniform_pos (r));
printf ("second value = %f\n", gsl_rng_uniform_pos (r));
```

```
  strcpy(filename,"Data/"); strcat(filename, project); strcat(filename,"_PseudoSnapShot.dump");

  datafile.open(filename,ios::trunc);

  datafile << "ITEM: TIMESTEP" << endl;
  datafile << "1" << endl;
  datafile << "ITEM: NUMBER OF ATOMS" << endl;
  datafile << Nmin-1 << endl;
  datafile << "ITEM: BOX BOUNDS" << endl;
  datafile << 0 << " " << sqrt(Area) << endl;
  datafile << 0 << " " << sqrt(Area) << endl;
  datafile << z[0] << " " << z[Nz-1] << endl;
  datafile << "ITEM: ATOMS id type xs ys zs" << endl;

//  datafile << "#" << setw(13) << "x (nm)" << setw(14) << "y (nm)" << setw(14) << "z (nm)" << setw(14) << "type" << endl;
  for (j=0;j<Nplus;j++) {
    datafile << j << " " << 1.5 + 0.5*fabs(zpos[j])/zpos[j] << " " << x[j] << " " << y[j] << " " << zpos[j] << endl;
  }
  for (j=Nplus;j<Nmin;j++) {
    datafile << j << " " << 3.5 + 0.5*fabs(zpos[j])/zpos[j] << " " << x[j] << " " << y[j] << " " << zpos[j] << endl;
  }
  datafile.close();


  strcpy(filename,"Data/"); strcat(filename, project); strcat(filename,"_PseudoSnapShot.mol2");

  datafile.open(filename,ios::trunc);

  datafile << "@<TRIPOS>MOLECULE" << endl;
  datafile << "system" << endl;
  datafile << Nmin-1 << " 0" << endl;
  datafile << "@<TRIPOS>ATOM" << endl;

//  datafile << "#" << setw(13) << "x (nm)" << setw(14) << "y (nm)" << setw(14) << "z (nm)" << setw(14) << "type" << endl;
  for (j=0;j<Nplus;j++) {
//    datafile << j << " N" << j << " " << x[j] << " " << y[j] << " " << zpos[j] << " C.1 " << 1.5 + 0.5*fabs(zpos[j])/zpos[j] << endl;
    datafile << j << " N" << j << " " << x[j] << " " << y[j] << " " << zpos[j] << " C.1 " << type[j] << endl;
  }
  for (j=Nplus;j<Nmin;j++) {
//    datafile << j << " N" << j << " " << x[j] << " " << y[j] << " " << zpos[j] << " C.1 " << 3.5 + 0.5*fabs(zpos[j])/zpos[j] << endl;
    datafile << j << " N" << j << " " << x[j] << " " << y[j] << " " << zpos[j] << " C.1 " << type[j] << endl;
  }
  datafile << "@<TRIPOS>BOND" << endl;
  datafile.close();

  gsl_rng_free (r); free (x); free (y); free (zpos);

}

void CoulombFluid::RecordCorrelationFunctions() {

  ofstream datafile;
  char filename[70];

  strcpy(filename,"Data/"); strcat(filename, project); strcat(filename,"_CorrelationFunctionsInt.dat");

  for (i=0;i<Nr;i++) { for (j=0;j<2*Nz;j++) {for (k=0;k<2*Nz;k++) {
    if (Qs[i][j][k]!=0.0) h[i][j][k] = -1.0 + 1e-10;
    //if (Qs[i][j][k]!=0.0) Qs[i][j][k] -= 2.0;
  }}}

  datafile.open(filename,ios::trunc);
  datafile << "#" << setw(19) << "1) r (nm)" << setw(20) << "2) z (nm)" << setw(20) << "3) h++(z_int)"<< setw(20) << "4) h+-(z_ph2)"<< setw(20) << "5) h+-(z_int)";
  datafile << setw(20) << "6) h++(zmin)"<< setw(20) << "7) h+-(zmin)"<< setw(20) << "8) c(z_int)" << setw(20) << "9) hMid phase1"<< setw(20) << "10) cMid phase1";
  datafile << setw(20) << "11) c(zmin)" << setw(20) << "12) hMid phase2"<< setw(20) << "13) cMid phase2" << setw(20) << "14) uRegHT_+-(k_int)";
  datafile << setw(20) << "15) uHT(kmin)" << setw(20) << "16) Qs(z_int)" << setw(20) << "17) (Qs+h)(z_int)" << setw(20) << "18) QsHT(z_int)";
  datafile << setw(20) << "19) h_+-h_-(z_int)" << setw(20) << "20) rho(z_int)" << setw(20) << "21) rho(ph2)" << setw(20) << "22) rho(ph1)";
  datafile << setw(20) << "23) rho(zmin)" << setw(20) << "24) h_hom" << setw(20) << "25) h_hom - h" << setw(20) << "26) uReg" << endl;
  for (i=Nr-1;i>=0;i--) { for (j=0;j<Nz;j++) {
    datafile << setw(20) << -r[i] << setw(20) << z[j] << setw(20) << h[i][j][Nz/2] << setw(20) << h[i][j][3*Nz/4 + Nz] << setw(20) << h[i][j][3*Nz/2]; //5
    datafile << setw(20) << h[i][j][0] << setw(20) << h[i][j][Nz] << setw(20) << c[i][j][3*Nz/2] << setw(20) << h[i][j][Nz/4] << setw(20) << c[i][j][Nz]; //10
    datafile << setw(20) << cm[j]*h[i][j][k]-cp[j]*h[i][j][Nz] << setw(20) << h[i][j][3*Nz/4] << setw(20) << cm[j]*(h[i][j][3*Nz/4]+1.0)-cp[j]*(h[i][j][3*Nz/4+Nz]+1.0) << setw(20) << uRegHT[i][j][Nz/2+3]; //14
    datafile << setw(20) << uHT[i][j][0] << setw(20) << Qs[i][j][Nz/2] << setw(20) << Qs[i][j][Nz/2]+h[i][j][Nz/2] << setw(20) << QsHT[i][j][Nz/2]; //18
    datafile << setw(20) << h[i][j][Nz/2]-h[i][j+Nz][3*Nz/2] << setw(20) << -cp[j]*(h[i][j][Nz/4]+1.0)+cm[j]*(h[i][j][Nz/4+Nz]+1.0); //20
    datafile << setw(20) << -cp[j]*(h[i][j][3*Nz/4]+1.0)+cm[j]*(h[i][j][3*Nz/4+Nz]+1.0) << setw(20) << -cp[j]*(h[i][j][Nz/2]+1.0)+cm[j]*(h[i][j][3*Nz/2]+1.0); //22
    datafile << setw(20) << -cp[j]*(h[i][j][0]+1.0)+cm[j]*(h[i][j][Nz]+1.0) << setw(20) << htest[i][j] << setw(20) << h[i][j][3*Nz/2]-Qs[i][j][3*Nz/2]-htest[i][j]; //25
    datafile << setw(20) << uReg[i][j][Nz/2] << setw(20) << uRegHT[i][j][Nz/2]; //27
    //datafile << setw(20) << (u[i][j][Nz/2]-u[i][j+Nz][3*Nz/2]) << setw(20) << (uReg[i][j][Nz/2]-uReg[i][j+Nz][3*Nz/2]) << setw(20) <<(uHT[i][j][Nz/2]-uHT[i][j+Nz][3*Nz/2]) << setw(20) << (uRegHT[i][j][Nz/2]-uRegHT[i][j+Nz][3*Nz/2]);
    datafile << endl;
  } datafile << endl;
  }
  for (i=0;i<Nr;i++) { for (j=0;j<Nz;j++) {
   datafile << setw(20) << r[i] << setw(20) << z[j] << setw(20) << h[i][j][Nz/2] << setw(20) << h[i][j][3*Nz/4 + Nz] << setw(20) << h[i][j][3*Nz/2];
    datafile << setw(20) << h[i][j][0] << setw(20) << h[i][j][Nz] << setw(20) << c[i][j][3*Nz/2] << setw(20) << h[i][j][Nz/4] << setw(20) << c[i][j][Nz];
    datafile << setw(20) << cm[j]*h[i][j][k]-cp[j]*h[i][j][Nz] << setw(20) << h[i][j][3*Nz/4] << setw(20) << cm[j]*(h[i][j][3*Nz/4]+1.0)-cp[j]*(h[i][j][3*Nz/4+Nz]+1.0) << setw(20) << uRegHT[i][j][Nz/2+3];
    datafile << setw(20) << uHT[i][j][0] << setw(20) << Qs[i][j][Nz/2] << setw(20) << Qs[i][j][Nz/2]+h[i][j][Nz/2] << setw(20) << QsHT[i][j][Nz/2];
    datafile << setw(20) << h[i][j][Nz/2]-h[i][j+Nz][3*Nz/2] << setw(20) << -cp[j]*(h[i][j][Nz/4]+1.0)+cm[j]*(h[i][j][Nz/4+Nz]+1.0);
    datafile << setw(20) << -cp[j]*(h[i][j][3*Nz/4]+1.0)+cm[j]*(h[i][j][3*Nz/4+Nz]+1.0) << setw(20) << -cp[j]*(h[i][j][Nz/2]+1.0)+cm[j]*(h[i][j][3*Nz/2]+1.0);
    datafile << setw(20) << -cp[j]*(h[i][j][0]+1.0)+cm[j]*(h[i][j][Nz]+1.0) << setw(20) << htest[i][j] << setw(20) << h[i][j][3*Nz/2]-Qs[i][j][3*Nz/2]-htest[i][j];
    datafile << setw(20) << uReg[i][j][Nz/2] << setw(20) << uRegHT[i][j][Nz/2];
    //datafile << setw(20) << (u[i][j][Nz/2]-u[i][j+Nz][3*Nz/2]) << setw(20) << (uReg[i][j][Nz/2]-uReg[i][j+Nz][3*Nz/2]) << setw(20) <<(uHT[i][j][Nz/2]-uHT[i][j+Nz][3*Nz/2]) << setw(20) << (uRegHT[i][j][Nz/2]-uRegHT[i][j+Nz][3*Nz/2]);
    datafile << endl;
  } datafile << endl;
  }
  datafile.close();

  ResetQs();

}

void CoulombFluid::RecordBulkCorrelationFunctions(double *s1, double *hp, double *hm, double *hpm, int N, int medium) {

  ofstream datafile;
  char filename[70];
```

```
  char *add,solvent[14];
  int length;

  strcpy(filename,"Data/"); strcat(filename, project); strcat(filename,"_BulkCorrelationFunctions");

  add = strstr (filename, "Functions");
  add += 9; sprintf(solvent,"_medium_%u.dat",medium); length = sizeof(solvent)/sizeof(char); strncpy (add,solvent,length+12);

  datafile.open(filename,ios::trunc);
  datafile << "# Correlation functions in reservoir " << medium << endl;
  datafile << "#" << setw(13) << "r (nm)" << setw(14) << "hp(r)"<< setw(14) << "hm(r)" << setw(14) << "hpm(r)"<< endl;
  for (i=0;i<N;i++) {
    datafile << setw(14) << s1[i] << setw(14) << hp[i] << setw(14) << hm[i] << setw(14) << hpm[i] << endl;
  }

  datafile.close();

}

void CoulombFluid::RecordTestfunctions() {

  ofstream datafile;
  char filename[70];

  strcpy(filename,"Data/"); strcat(filename, project); strcat(filename,"_TestfunctionsPL.dat"); /* "parallel" */

  datafile.open(filename,ios::trunc);
//    datafile << "#" << setw(13) << "r (nm)" << setw(14) << "ck" << setw(14) << "hk" << setw(14) << "Qsk" << setw(14) << "uRegk" << setw(14) << "uk" << endl;
  datafile << "#" << setw(13) << "r (nm)" << setw(14) << "htest" << setw(14) << "hcenter+-" << setw(14) << "hint+-" << setw(14) << "hcenter++" << setw(14) << "hint++" << endl;
  for (i=0;i<Nr;i++) {
//    datafile << setw(14) << kFT[i] << setw(14) << c[i][3*Nz/2][Nz/2] << setw(14) << newh[i][3*Nz/2][Nz/2] << setw(14) << QsHT[i][3*Nz/2][Nz/2] << setw(14) << uRegHT[i][3*Nz/2][Nz/2] << setw(14) << uHT[i][3*Nz/2][Nz/2] << endl;
    datafile << setw(14) << r[i] << setw(14) << htest[i][Nz/2] << setw(14) << h[i][3*Nz/2][Nz/2]-Qs[i][3*Nz/2][Nz/2] << setw(14) << h[i][Nz][0]-Qs[i][Nz][0] << setw(14) << h[i][Nz/2][Nz/2]-Qs[i][Nz/2][Nz/2] << setw(14) << h[i][0][0]-Qs[i][0][0] << setw(14) << kappa1*r[i] << endl;
  }
  datafile.close();

  strcpy(filename,"Data/"); strcat(filename, project); strcat(filename,"_TestfunctionsPP.dat"); /* "perpendicular" */

  datafile.open(filename,ios::trunc);
//    datafile << "#" << setw(13) << "r (nm)" << setw(14) << "ck" << setw(14) << "hk" << setw(14) << "Qsk" << setw(14) << "uRegk" << setw(14) << "uk" << endl;
  datafile << "#" << setw(13) << "z (nm)" << setw(14) << "htest" << setw(14) << "hcenter" << setw(14) << "hint" << endl;
  for (j=Nz/2;j<Nz;j++) {
    datafile << setw(14) << z[j] << setw(14) << htest[0][j] << setw(14) << h[0][3*Nz/2][j]-Qs[0][3*Nz/2][j] << setw(14) << h[0][Nz][j-Nz/2]-Qs[0][Nz][j-Nz/2] << endl;
  }
  datafile.close();

}

void CoulombFluid::RecordConvolutionFunctions(double *ThetaConv) {

  ofstream datafile;
  char filename[70];

  strcpy(filename,"Data/"); strcat(filename, project); strcat(filename,"_ConvolutionFunctions.dat"); /* "parallel" */

  datafile.open(filename,ios::trunc);
//    datafile << "#" << setw(13) << "r (nm)" << setw(14) << "ck" << setw(14) << "hk" << setw(14) << "Qsk" << setw(14) << "uRegk" << setw(14) << "uk" << endl;
  datafile << "#" << setw(13) << "r (nm)" << setw(14) << "hcenter+-" << setw(14) << "h_conv+-" << setw(14) << "theta" << setw(14) << "theta_conv" << setw(14) << "hmax" << setw(14) << "h_conv_max" << endl;
  for (i=0;i<Nr;i++) {
//    datafile << setw(14) << kFT[i] << setw(14) << c[i][3*Nz/2][Nz/2] << setw(14) << newh[i][3*Nz/2][Nz/2] << setw(14) << QsHT[i][3*Nz/2][Nz/2] << setw(14) << uRegHT[i][3*Nz/2][Nz/2] << setw(14) << uHT[i][3*Nz/2][Nz/2] << endl;
    datafile << setw(14) << r[i] << setw(14) << h[i][3*Nz/2][Nz/2] << setw(14) << newg[i][3*Nz/2][Nz/2] << setw(14) << Theta[i] << setw(14) << ThetaConv[i] << setw(14) << h[i][jmax][kmax] << setw(14) << newg[i][jmax][kmax] << endl;
  }
  datafile.close();

}

void CoulombFluid::RecordTime (int *Time) {

  ofstream datafile;
  char filename[50];

  strcpy(filename,"Data/"); strcat(filename, project); strcat(filename,"_Info.dat");

  datafile.open(filename,ios::app);
  datafile << "\n" << setw(10) << evaluations << setw(40) << " evaluations before convergence.\n" << endl;
  datafile << "\nprogram ran for " << setprecision (0) << Time[0] << " hours, " << Time[1] << " minutes, and " << Time[2] << " seconds" << endl;
  datafile.close();

  cout << "\nprogram ran for " << setprecision (0) << Time[0] << " hours, " << Time[1] << " minutes, and " << Time[2] << " seconds" << endl;

}

void CoulombFluid::RecordFreeEnergy(bool append) {

  ofstream datafile;
  char filename[70];
  int nnf = nf-1;

  strcpy(filename,"Data/"); strcat(filename, project); strcat(filename,"_FreeEnergy.dat");
  cout << "TEST: nf = " << nf << endl;
  if (!append) {
    datafile.open(filename,ios::trunc);
    datafile << "#" << setw(29) << "1) H (1/kappa1)" << setw(30) << "2) Fid / kBT" << setw(30) << "3) Fext / kBT";
    datafile << setw(30) << "4) Fexc / kBT" << setw(30) << "5) Ftot / H kBT" << setw(30) << "6) (Ftot-Fb) / H kBT" << setw(30) << "7) (Ftot - Fb) / kBT";
    datafile << setw(30) << "8) Z1Z1 / A" << setw(30) << "9) Z1Z2 / A" << setw(30) << "10) Z2Z2-NP-NM/ A";
    datafile << setw(30) << "11) N+ / A" << setw(30) << "12) N- / A" << setw(30) << "13) Pzz" << endl;
    for (j=0;j<nf;j++) {
      datafile << setw(30) << Hf[j]*kappa1 << setw(30) << Fid[j] << setw(30) << Fext[j] << setw(30) << Fexc[j];
      datafile << setw(30) << (Fid[j]+Fext[j]+Fexc[j])/Hf[j] << setw(30) << (Fid[j]+Fext[j]+Fexc[j])/Hf[j]-(Fid[0]+Fexc[0])/Hf[0];
      datafile << setw(30) << ((Fid[j]+Fext[j]+Fexc[j])/Hf[j]-(Fid[0]+Fexc[0])/Hf[0])*Hf[j];
      datafile << setw(30) << Z1Z1[j] + NP[j] + NM[j] << setw(30) << Z1Z2[j] << setw(30) << Z2Z2[j];
      datafile << setw(30) << NP[j] << setw(30) << NM[j] << setw(30) << Pzz[j] << endl;
    }
  } else {
    datafile.open(filename,ios::app);
    datafile << setw(30) << Hf[nnf]*kappa1 << setw(30) << Fid[nnf] << setw(30) << Fext[nnf] << setw(30) << Fexc[nnf];
```

```
    datafile << setw(30) << (Fid[nnf]+Fext[nnf]+Fexc[nnf])/Hf[nnf] << setw(30) << (Fid[nnf]+Fext[nnf]+Fexc[nnf])/Hf[nnf]-(Fid[0]+Fexc[0])/Hf[0];
    datafile << setw(30) << ((Fid[nnf]+Fext[nnf]+Fexc[nnf])/Hf[nnf]-(Fid[0]+Fexc[0])/Hf[0])*Hf[nnf];
    datafile << setw(30) << Z1Z1[nnf] + NP[nnf] + NM[nnf] << setw(30) << Z1Z2[nnf] << setw(30) << Z2Z2[nnf];
    datafile << setw(30) << NP[nnf] << setw(30) << NM[nnf] << setw(30) << Pzz[nnf] << endl;
  }
  datafile.close();

}

int test_dht_exp1(void) {


  /* DHT transform from r to k-space missed a factor 2*pi ( f(k=0) should be equal to the integral over f(r); it was 2*pi too low )
  The k-coordinate should be scaled with 1/(2*pi) to get the correct value for f(r=0)  */

  int stat = 0;
  int n,N=100;
  double *f_in, *f_out, *f_id,*kFT,*xFT,x,k,I=100.0;
  double Delta1,Delta2,a,b,sum;
  gsl_dht * t = gsl_dht_new(N, 0.0, I);
  ofstream datafile;

  gsl_dht * t2 = gsl_dht_new(N, 0.0, gsl_dht_k_sample(t, N));

  f_in = (double*) calloc (N,sizeof(double));
  f_out = (double*) calloc (N,sizeof(double));
  f_id = (double*) calloc (N,sizeof(double));
  kFT = (double*) calloc (N+1,sizeof(double));
  xFT = (double*) calloc (N+1,sizeof(double));

  for (n=0;n<N+1;n++) { kFT[n] = gsl_dht_k_sample(t, n); xFT[n] = gsl_dht_x_sample(t, n); }

  Delta1 = 10.0; Delta2 = -5.0; a = 20.0; b = gsl_sf_bessel_zero_J0(N+1)/I/I;

  for(n=0; n<N; n++) {
    x = gsl_dht_x_sample(t, n);  k = gsl_dht_k_sample(t, n);
//    f_in[n] = 1.0/x; //exp(-0.25*x);
//    if (x<5.0) f_in[n] = 1.0; else f_in[n] = 0.0;
//    if (x=0.0) f_in[n] = 1e30/(2*pi*x); else f_in[n] = 0.0;
//    if (x<0.0) f_in[n] = 0.0;
//    f_in[n] = 1.0/sqrt(2.0/pi + x*x); //becomes a yukawa function //exp(-2*x)/x;
//    f_in[n] = Delta1 + 0.5*(x*x-a*a)*Delta2/a;
//    if (x > a) f_in[n] = Delta1*exp((x-a)*Delta2/Delta1);     else f_in[n] = 0.0;
//    f_in[n] = 1.0/sqrt(a*a + x*x); //becomes a yukawa function //exp(-a*x)/x;
//    f_in[n] = exp(-a*x*b)/x; // inverse DHT of above function;
    if (x < a) f_in[n] = Delta1 + 0.5*(x*x-a*a)*Delta2/a; //"parabolic block function"
//    f_in[n] = a*(Delta1*gsl_sf_bessel_J1(a*k)/k - Delta2*gsl_sf_bessel_Jn(2,a*k)/k/k); //DHT of the "parabolic block function";
  }

  gsl_dht_apply(t, f_in, f_out);
  gsl_dht_apply(t2, f_out, f_id);

  cout << "b = " << b << ", b*I*I/N = " << b*I*I/N << " and b*I = " << b*I << endl;
  cout << "Imax = " << xFT[N] << " and kmax = " << kFT[N] << ". Ratio = " << xFT[N]/kFT[N] << endl;

//  for(n=0;n<N;n++) f_id[n] /= gsl_sf_bessel_J0 (20.0);
/*â€" Function: double gsl_sf_bessel_zero_J0 (unsigned int s)
â€" Function: int gsl_sf_bessel_zero_J0_e (unsigned int s, gsl_sf_result * result)

    These routines compute the location of the s-th positive zero of the Bessel function J_0(x).
*/

  sum = 0;
  for (n=0;n<N-1;n++) {
    sum += (f_out[n]*2*pi) *2*pi*kFT[n]/(2*pi) * (kFT[n+1]-kFT[n])/(2*pi);
  }

  cout << "f(0) = " << f_in[0] << " and k-integral gives " << sum << ". The ratio of the two is " << sum/f_in[0] << endl;

  sum = 0;
  for (n=0;n<N-1;n++) {
    sum += f_in[n]*2*pi*xFT[n]*(xFT[n]-xFT[n-1]);
  }

  cout << "fFT(0) = " << f_out[0]*2*pi << " and x-integral gives " << sum << ". The ratio of the two is " << sum/f_out[0] << endl;

  sum = 0;
  for (n=0;n<N-1;n++) {
    sum += f_in[n]-f_id[n];
  }

  cout << " Total absolute difference after one iteration is " << sum << endl;

  datafile.open("Data/testdata.dat");

  datafile << "\n#" << setw(20) << "r" << setw(20) << "k" << setw(20) << "f(r)" << setw(20) << "fDHT(k)" << setw(20) << "fDHTDHT(r)" << endl;

  for (n=0;n<N;n++) {
    x = gsl_dht_x_sample(t, n); k = gsl_dht_k_sample(t, n);//pow(gsl_sf_bessel_zero_J0(N),2.0)* f_in[n]/f_id[n]
    datafile << setw(20) << x << setw(20) << k << setw(20) << f_in[n] << setw(20) << f_out[n] << setw(20) << f_id[n];
//    datafile << setw(20) << exp(-2.0*x)/x;
    datafile << setw(20) << exp(-a*k)/k;
    datafile << setw(20) << a*(Delta1*gsl_sf_bessel_J1(a*k)/k - Delta2*gsl_sf_bessel_Jn(2,a*k)/k/k);
    datafile << setw(20) << 5.0  *  gsl_sf_bessel_J1(5*pi*k)/k << setw(20) << b << setw(20) << f_in[n]/f_out[n]/b << setw(20) << exp(-a*k);
    datafile << endl;
  }

  datafile.close();



  gsl_dht_free(t);

  return stat;
```

```
}

void CoulombFluid::test_dht_potentials() {

  SetGrid();
  SetPairPotential2Walls();
//  SetHankelTransformPairPotential();
  EvaluateReservoirMuEx(1);
  ResetQs();

  double I = r[Nr-1];
  gsl_dht * t = gsl_dht_new(Nr, 0.0, I);
  Mat3D_DP uTestHT;
  double *uvec, *uvecHT;
  ofstream datafile;
  char filename[70];

  uvec = (double*) calloc (Nr,sizeof(double));  uvecHT = (double*) calloc (Nr,sizeof(double));

  uTestHT.assign(0.0,Nr,2*Nz,2*Nz);

  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
      for (i=0;i<Nr;i++) { uvec[i] = u[i][j][k]*exp(-3*pow(r[i]/r[Nr-1],2.0)); }
      gsl_dht_apply(t, uvec, uvecHT);
      for (i=0;i<Nr;i++) { uTestHT[i][j][k] = uvecHT[i];}//-uHT[i][j][k]; }
  }}

/*
  for (j=0;j<2*Nz;j++){      for (k=0;k<2*Nz;k++){
      for (i=0;i<Nr;i++) { uvec[i] = uHT[i][j][k]; }
      gsl_dht_apply(t, uvec, uvecHT); for (i=0;i<Nr;i++) uvecHT[i] *= gsl_sf_bessel_zero_J0(Nr+1)/I/I;
      for (i=0;i<Nr;i++) { uTestHT[i][j][k] = uvecHT[i];}//-uHT[i][j][k]; }
  }}
*/

  cout << "(Nr-1)^th sample point in r-space " << gsl_dht_x_sample (t,Nr-1) << endl;
  cout << "(Nr-1)^th sample point in k-space " << gsl_dht_k_sample (t,Nr-1) << endl;
  cout << "fraction " << gsl_dht_x_sample (t,Nr-1)/gsl_dht_k_sample (t,Nr-1) << endl;

  strcpy(filename,"Data/"); strcat(filename, project); strcat(filename,"_TestPotential.dat");

  datafile.open(filename,ios::trunc);
  datafile << "#" << setw(13) << "r (nm)" << setw(14) << "u(z_int)"<< setw(14) << "uHT(z_int)" << setw(14) << "uTestHT(z_int)" << setw(14) << "uMid phase1"<< setw(14) << "uHTMid phase1" << setw(14) << "uTestHTMid phase1" << endl;
  for (i=0;i<Nr;i++) {
    datafile << setw(14) << r[i]/gsl_dht_x_sample (t,Nr-1)*gsl_dht_k_sample (t,Nr-1) << setw(14) << Qs[i][Nz/2][Nz/2] << setw(14) << QsHT[i][Nz/2][Nz/2] << setw(14) << uTestHT[i][Nz/2][Nz/2] << setw(14) << u[i][Nz/4+4][Nz/4] << setw(14) << uHT[i][Nz/4+4][Nz/4]*gsl_dht_k_sample (t,Nr-
  }

  datafile.close();

}

void CoulombFluid::TestReservoir() {

  int m,M=50;
  ofstream datafile;
  char filename[70];

  strcpy(filename,"Data/"); strcat(filename, project); strcat(filename,"_ChemicalPotentials.dat");

  datafile.open(filename,ios::trunc);
  datafile << "#" << setw(13) << "cs1 (M)" << setw(14) << "mu1+ (kT)" << setw(14) << "mu1- (kT)" << setw(14) << "mu2+ (kT)" << setw(14) << "mu2- (kT)" << endl;

  cout << setw(14) << "cs1 (M)" << setw(14) << "mu1+ (kT)" << setw(14) << "mu1- (kT)" << setw(14) << "mu2+ (kT)" << setw(14) << "mu2- (kT)" << endl;

  qp = 2.0;
  qm = 2.0;
  for (m=0;m<M;m++) {
    cs1 = pow(10.0,-6.0 + m*5.0/(M-1.0));
    kappa1 = sqrt(8*pi*lB1*qp*cs1);
    cs2 = qp/qm*cs1*exp(-0.25*(qp*qp/ap+qm*qm/am)*(lB2-lB1));
    kappa2 = sqrt(8*pi*lB2*qp*cs2);

    EvaluateReservoirMuEx(1);
//    EvaluateReservoirMuEx(2);

    datafile << setw(14) << cs1 << setw(14) << muEx1p << setw(14) << muEx1m << setw(14) << muEx2p << setw(14) << muEx2m << endl;

    cout << setw(14) << cs1 << setw(14) << muEx1p << setw(14) << muEx1m << setw(14) << muEx2p << setw(14) << muEx2m << endl;

  }
  datafile.close();

}

void CoulombFluid::TestMatrixInversion() {

  Nz = 12;
  double sum,det;

  gsl_matrix * m = gsl_matrix_alloc (Nz, Nz),* mLU = gsl_matrix_alloc (Nz, Nz), * m2 = gsl_matrix_alloc (Nz, Nz), * Id = gsl_matrix_alloc (Nz, Nz);//,* mmem = gsl_matrix_alloc (Nz, Nz);
  gsl_vector * hvec = gsl_vector_alloc (Nz), * cvec = gsl_vector_alloc (Nz);//, * res = gsl_vector_alloc (Nz)
  gsl_permutation * p = gsl_permutation_alloc (Nz);
  int sign[1];


//  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){ gsl_matrix_set (m, j, k, j + k + j*k + log(k+1));}}// cout << gsl_matrix_get (m, j, k) << endl; }}
//  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){ gsl_matrix_set (m, j, k, (j+k + j*k*k*j+ log(k+5) + log(j+5))/30 );}}
  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){ gsl_matrix_set (m, j, k, j*Nz+k+1);}}
  for (j=0;j<Nz;j++){      for (k=j;k<Nz;k++){ gsl_matrix_set (m, j, k, gsl_matrix_get(m,k,j));}}
//  for (j=1;j<Nz-1;j++){ gsl_matrix_set (m, j, j, j+2);}

  cout << "\nMatrix M :\n" << endl;
  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    cout << setw(10) << gsl_matrix_get(m,j,k);
  } cout << endl; }
```

```
  gsl_matrix_memcpy(mLU,m);
  gsl_linalg_LU_decomp(mLU,p,sign);
//  gsl_linalg_cholesky_decomp(mLU);  gsl_linalg_cholesky_invert(mLU);  gsl_matrix_memcpy(m2,mLU);

  cout << "\nLU decomposition of M :\n" << endl;
  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    cout << setw(10) << setprecision(2) << gsl_matrix_get(mLU,j,k);
  } cout << endl; }

//  gsl_linalg_LU_invert(mLU,p,m2);
//  gsl_matrix_memcpy(mLU,m);
//  det = exp(gsl_linalg_LU_lndet(mLU)); cout << "Determinant of M = " << det << endl;
  det = gsl_linalg_LU_det(mLU,*sign); cout << "Determinant of M = " << det << endl;
//  gsl_matrix_scale (mLU, 1.0/det);
  for (k=0;k<Nz;k++){
    for (j=0;j<Nz;j++) gsl_vector_set (cvec,j,delta[j][k]);
//    gsl_matrix_memcpy(mmem,m);    gsl_linalg_LU_refine(mmem,mLU,p,cvec,hvec,res);
//   gsl_linalg_LU_svx(mLU,p,cvec);
//    gsl_linalg_LU_invert(mLU,p,m2);
//    gsl_linalg_LU_refine(m,mLU,p,cvec,hvec,res);
//    gsl_linalg_LU_refine(m,mLU,p,cvec,hvec,res);
//    gsl_matrix_memcpy(mLU,m);    gsl_linalg_HH_svx(mLU,cvec);
    gsl_linalg_LU_solve(mLU,p,cvec,hvec);
    for (j=0;j<Nz;j++) gsl_matrix_set (m2,j,k,gsl_vector_get (hvec,j));
  }

  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    if (fabs(gsl_matrix_get(m2,j,k)) < 1e-12) gsl_matrix_set(m2,j,k,0.0);
  }}

  cout << "\nM inverse :\n" << endl;
  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    cout << setw(10) << setprecision(2) << gsl_matrix_get(m2,j,k);
  } cout << endl; }

  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    sum = 0; for (i=0;i<Nz;i++) sum += gsl_matrix_get(m,j,i)*gsl_matrix_get(m2,i,k);
    gsl_matrix_set(Id,j,k,sum);
  }}

  cout << "\nM times its inverse (should be the identity matrix) :\n" << endl;
  for (j=0;j<Nz;j++){      for (k=0;k<Nz;k++){
    cout << setw(10) << setprecision(2) << gsl_matrix_get(Id,j,k);
  } cout << endl; }


}

void CoulombFluid::TestConvolution(){

  //  ResetValues();
  SetGrid2Walls1Dielectric();

  double *Ftest,*Fconv,*FtestFTconv,*FtestFT,*Ffinal,*Fdirect,*Aux, *Fconv2nd,*FconvFT2nd;
  double I = r[Nr-1],b,d, *ThetaConv, *ThetaTest, *ThetaTestFT, thetaMax,dfactor=4.0;
  gsl_dht * t = gsl_dht_new(Nr, 0.0, I);
  gsl_permutation * p = gsl_permutation_alloc (2*Nz);

  Ftest = (double*) calloc (Nr,sizeof(double));  Fconv = (double*) calloc (Nr,sizeof(double));  FtestFTconv = (double*) calloc (Nr,sizeof(double));
  FtestFT = (double*) calloc (Nr,sizeof(double));  Ffinal = (double*) calloc (Nr,sizeof(double));  Fdirect = (double*) calloc (Nr,sizeof(double));
  Aux = (double*) calloc (Nr,sizeof(double)); Fconv2nd = (double*) calloc (Nr,sizeof(double)); FconvFT2nd = (double*) calloc (Nr,sizeof(double));
  ThetaConv = (double*) calloc (Nr,sizeof(double)); ThetaTest = (double*) calloc (Nr,sizeof(double)); ThetaTestFT = (double*) calloc (Nr,sizeof(double));

  b = gsl_sf_bessel_zero_J0(Nr+1)/I/I;

//  cout << "I = " << I << " and j0(Nr+1) = " << gsl_sf_bessel_zero_J0(Nr+1) << endl;
//  cout << "k(Nr) = " << gsl_dht_k_sample(t, Nr) << " and r(Nr) = " << gsl_dht_x_sample(t, Nr) << endl;

  i=0; while (r[i]<I/dfactor) i++; j=i-1;
  for (i=0;i<Nr;i++) {
    if (r[i]>2*ap) Ftest[i] = 8*exp(-(r[i]-2*ap)*25.0/(r[Nr-1]-2*ap));
    //if (r[i]>=0*ap) Ftest[i] = exp(-8*r[i]*r[i]*pi);
    //if (r[i]>=0*ap and i<=j) Ftest[i] = exp(-0.003*dfactor*dfactor*r[i]*r[i]*pi); if (i>j) Ftest[i] = Ftest[j]*exp((Ftest[j]-Ftest[j-1])/(r[j]-r[j-1])*(r[i]-r[j])/Ftest[j]);
    //if (r[i]>2*ap) Ftest[i] = 8*exp(-(r[i]-2*ap)*25.0/(r[Nr-1]-2*ap)) / r[i];
    if (r[i]<period) ThetaTest[i]=1.0;
  }

  // The convolution of Ftest with Theta_d(r)
  for (i=1;i<Nr;i++) {
    Fconv[i] = 0;
    for (j=1;j<Nr;j++) {
      d =  period*period - pow(r[j]-r[i],2.0);
      if ( period*period - pow(r[j]+r[i],2.0) > 0 ) thetaMax = 2*pi; else thetaMax = 2*acos( 1.0 - d/(2.0*r[j]*r[i]) );
      if (d>0) Fconv[i] += (r[j]-r[j-1])*r[j]*Ftest[j]*thetaMax;
    }
  }
  Fconv[0] = Fconv[1];

  gsl_dht_apply(t, Theta, ThetaConv); for (i=0;i<Nr;i++) ThetaConv[i] *= b*b/(2*pi);

  gsl_dht_apply(t, Fconv, FtestFTconv); for (i=0;i<Nr;i++) FtestFTconv[i] *= 2*pi;

  for (i=0;i<Nr;i++) { FtestFT[i] = FtestFTconv[i]/Theta[i];  }

  gsl_dht_apply(t, FtestFT, Ffinal); for (i=0;i<Nr;i++) Ffinal[i] *= b*b/(2*pi);

  gsl_dht_apply(t, Ftest, Aux); for (i=0;i<Nr;i++) Aux[i] *= 2*pi;
  gsl_dht_apply(t, Aux, Fdirect); for (i=0;i<Nr;i++) Fdirect[i] *= b*b/(2*pi);
  gsl_dht_apply(t, ThetaTest, ThetaTestFT); for (i=0;i<Nr;i++) ThetaTestFT[i] *= 2*pi;

  for (i=0;i<Nr;i++) { FconvFT2nd[i] = Aux[i]*Theta[i];  }
  gsl_dht_apply(t, FconvFT2nd, Fconv2nd); for (i=0;i<Nr;i++) Fconv2nd[i] *= b*b/(2*pi);

  ofstream datafile;
```

```
  char filename[70];

  strcpy(filename,"Data/"); strcat(filename, project); strcat(filename,"_TestConvolution.dat");

  datafile.open(filename,ios::trunc);
  datafile << "#" << setw(29) << "1) r" << setw(30) << "2) k" << setw(30) << "3) Ftest(r)" << setw(30) << "4) Fconv(r)" << setw(30) << "5) FtestFTconv(k)";
  datafile << setw(30) << "6) FtestFT(k)" << setw(30) << "7) Ffinal(r)" << setw(30) << "8) Theta_d(r)" << setw(30) << "9) Theta_dFT(k)";
  datafile << setw(30) << "10) ThetaTest(r)" << setw(30) << "11) ThetaTestFT(k)" << setw(30) << "12) Fdirect(r)" << setw(30) << "13) Aux(k)";
  datafile << setw(30) << "14) Fconv2nd(r)" << setw(30) << "15) FconvFT2nd(k)" << endl;
  for (i=0;i<Nr;i++) {
    datafile << setw(30) << r[i] << setw(30) << kFT[i] << setw(30) << Ftest[i] << setw(30) << Fconv[i] << setw(30) << FtestFTconv[i];
    datafile << setw(30) << FtestFT[i] << setw(30) << Ffinal[i] << setw(30) << ThetaConv[i] << setw(30) << Theta[i];
    datafile << setw(30) << ThetaTest[i] << setw(30) << ThetaTestFT[i] << setw(30) << Fdirect[i] << setw(30) << Aux[i];
    datafile << setw(30) << Fconv2nd[i] << setw(30) << FconvFT2nd[i] << endl;
  }
  datafile.close();



  free(Ftest); free(Fconv); free(FtestFTconv); free(FtestFT); free(Ffinal); free(Fdirect); free(ThetaConv); free(ThetaTest); free(ThetaTestFT); free(Aux);
  gsl_dht_free (t); gsl_permutation_free (p);

}

void CoulombFluid::TestGaussianConvolution(){

  ResetValues(); SetGrid2Walls1Dielectric();

  double *Ftest,*Fconv,*FtestFTconv,*FtestFT,*Ffinal,*Fdirect,*Aux, *Fconv2nd,*FconvFT2nd;
  double I = r[Nr-1],b,d, *ThetaConv, *ThetaTest, *ThetaTestFT, thetaMax;
  gsl_dht * t = gsl_dht_new(Nr, 0.0, I);
  gsl_permutation * p = gsl_permutation_alloc (2*Nz);

  Ftest = (double*) calloc (Nr,sizeof(double));  Fconv = (double*) calloc (Nr,sizeof(double));  FtestFTconv = (double*) calloc (Nr,sizeof(double));
  FtestFT = (double*) calloc (Nr,sizeof(double));  Ffinal = (double*) calloc (Nr,sizeof(double));  Fdirect = (double*) calloc (Nr,sizeof(double));
  Aux = (double*) calloc (Nr,sizeof(double)); Fconv2nd = (double*) calloc (Nr,sizeof(double)); FconvFT2nd = (double*) calloc (Nr,sizeof(double));
  ThetaConv = (double*) calloc (Nr,sizeof(double)); ThetaTest = (double*) calloc (Nr,sizeof(double)); ThetaTestFT = (double*) calloc (Nr,sizeof(double));

  b = gsl_sf_bessel_zero_J0(Nr+1)/I/I;

//  cout << "I = " << I << " and j0(Nr+1) = " << gsl_sf_bessel_zero_J0(Nr+1) << endl;
//  cout << "k(Nr) = " << gsl_dht_k_sample(t, Nr) << " and r(Nr) = " << gsl_dht_x_sample(t, Nr) << endl;

  for (i=0;i<Nr;i++) {
//    if (r[i]>8*ap) Ftest[i] = 8*exp(-(r[i]-2*ap)*5.0/(r[Nr-1]-2*ap));
    if (r[i]>=0*ap) Ftest[i] = exp(-8*r[i]*r[i]*pi);
//    if (r[i]>2*ap) Ftest[i] = 8*exp(-(r[i]-2*ap)*25.0/(r[Nr-1]-2*ap)) / r[i];
    if (r[i]<period) ThetaTest[i]=exp(-pi*period*r[i]*r[i]);
  }

  // The convolution of Ftest with Theta_d(r)
  for (i=1;i<Nr;i++) {
    Fconv[i] = 0;
    for (j=1;j<Nr;j++) {
      Fconv[i] += (r[j]-r[j-1])*r[j]*Ftest[j]*exp(-pi*period*(r[i]*r[i]+r[j]*r[j])) * 2*pi*gsl_sf_bessel_I0(period*pi*r[i]*r[j]);
    }
  }
  Fconv[0] = Fconv[1];

  gsl_dht_apply(t, ThetaGauss, ThetaConv); for (i=0;i<Nr;i++) ThetaConv[i] *= b*b/(2*pi);

  gsl_dht_apply(t, Fconv, FtestFTconv); for (i=0;i<Nr;i++) FtestFTconv[i] *= 2*pi;

  for (i=0;i<Nr;i++) { FtestFT[i] = FtestFTconv[i]/ThetaGauss[i];  }

  gsl_dht_apply(t, FtestFT, Ffinal); for (i=0;i<Nr;i++) Ffinal[i] *= b*b/(2*pi);

  gsl_dht_apply(t, Ftest, Aux); for (i=0;i<Nr;i++) Aux[i] *= 2*pi;
  gsl_dht_apply(t, Aux, Fdirect); for (i=0;i<Nr;i++) Fdirect[i] *= b*b/(2*pi);
  gsl_dht_apply(t, ThetaTest, ThetaTestFT); for (i=0;i<Nr;i++) ThetaTestFT[i] *= 2*pi;

  for (i=0;i<Nr;i++) { FconvFT2nd[i] = Aux[i]*ThetaGauss[i];  }
  gsl_dht_apply(t, FconvFT2nd, Fconv2nd); for (i=0;i<Nr;i++) Fconv2nd[i] *= b*b/(2*pi);

  ofstream datafile;
  char filename[70];

  strcpy(filename,"Data/"); strcat(filename, project); strcat(filename,"_TestGaussianConvolution.dat");

  datafile.open(filename,ios::trunc);
  datafile << "#" << setw(29) << "1) r" << setw(30) << "2) k" << setw(30) << "3) Ftest(r)" << setw(30) << "4) Fconv(r)" << setw(30) << "5) FtestFTconv(r)";
  datafile << setw(30) << "6) FtestFT(r)" << setw(30) << "7) Ffinal(r)" << setw(30) << "8) Theta_d(r)" << setw(30) << "9) Theta_dFT(r)";
  datafile << setw(30) << "10) ThetaTest(r)" << setw(30) << "11) ThetaTestFT(r)" << setw(30) << "12) Fdirect(r)" << setw(30) << "13) Aux(r)";
  datafile << setw(30) << "14) Fconv2nd(r)" << setw(30) << "15) FconvFT2nd(r)" << endl;
  for (i=0;i<Nr;i++) {
    datafile << setw(30) << r[i] << setw(30) << kFT[i] << setw(30) << Ftest[i] << setw(30) << Fconv[i] << setw(30) << FtestFTconv[i];
    datafile << setw(30) << FtestFT[i] << setw(30) << Ffinal[i] << setw(30) << ThetaConv[i] << setw(30) << ThetaGauss[i];
    datafile << setw(30) << ThetaTest[i] << setw(30) << ThetaTestFT[i] << setw(30) << Fdirect[i] << setw(30) << Aux[i];
    datafile << setw(30) << Fconv2nd[i] << setw(30) << FconvFT2nd[i] << endl;
  }
  datafile.close();



  free(Ftest); free(Fconv); free(FtestFTconv); free(FtestFT); free(Ffinal); free(Fdirect); free(ThetaConv); free(ThetaTest); free(ThetaTestFT); free(Aux);
  gsl_dht_free (t); gsl_permutation_free (p);

}

void CoulombFluid::ObtainDisjoiningPressure(){

  FILE *fp;
  double par[20], *Posm, *PosmV, *Width, previous, previousRho, FormerPhi, DeltaV=0,Press=0,zInit,Vlength=1e10;
```

```
char str[40], filename[80];
int Nmax = 1e7,Plength,Ncolumn=20, start;
bool DeltaVEnd=false;
ofstream datafile;

Posm = (double*) calloc (Nmax, sizeof(double));
PosmV = (double*) calloc (Nmax, sizeof(double));
Width = (double*) calloc (Nmax, sizeof(double));

strcpy(filename,"Data/"); strcat(filename, project); strcat(filename,"_Profiles.dat");

fp = fopen(filename, "r");

cout << "The input file is " << filename << endl;

if (fp == NULL) {
  fprintf(stderr, "Can't open input file %s\n",filename);
  exit(1);
}
while (!feof(fp)) {
  fscanf(fp, "%s", str);
  if (strcmp(str,"uRegInt") == 0) break;
}
start = ftell(fp);
/*
while (i<Ncolumn*Nmax-1 and (strcmp(str,"nan") != 0) and !feof(fp)) {
  fscanf(fp, "%s", str);
  cout << "reading i = " << i << " " << str << endl;
  i++;
}  Nmax = i/Ncolumn; cout << "Nmax = " << Nmax << endl;
fseek(fp,start,SEEK_SET);
*/
i = 0; j = 0; previous=-1; FormerPhi = 1e1;
while (i<Nmax-1 and !feof(fp)) {
  if (i%100 == 0) cout << "reading i = " << i << endl;
  if (fscanf(fp, "%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf ", &par[0],&par[1],&par[2], &par[3],&par[4],&par[5], &par[6],&par[7],&par[8], &par[9],&par[10],&par[11], &par[12],&par[13],&par[14], &par[15],&par[16],&par[17], &par[18],&par[19]) != 2
    cout << "Unexpected number of columns. Code is scanning the file " << filename << " for 20 columns." << endl;
    break;
  }
  //Press += (p[12]-Vprevious)*(par[5]+par[6]);

  if (i==1) { DeltaV = par[12]; zInit = previous; }    if (i==2) { DeltaV = (par[12]-DeltaV)/(par[0]-previous); Vlength = 1.0/DeltaV*(par[0]-previous); }
  if (par[0]-zInit > Vlength and !DeltaVEnd) { DeltaVEnd = true; Press += -DeltaV*previousRho*(Vlength-previous); }

  if (!DeltaVEnd) { Press += -DeltaV*(previousRho)*(par[0]-previous); }//cout << "Press = " << Press << " at i = " << i << " and Vlength = " << Vlength << endl; }
  //cout << "DeltaV = " << DeltaV << endl;

  if( (fabs(par[0])<1e-6 or ((fabs(par[0] + previous) < 1e-6) and previous < 0))  or  (fabs(par[8])<FormerPhi) ) { Posm[j] = (par[5]+par[6])*NA*kB*T*1e21; }
  if((par[0] < previous) or feof(fp)) { Width[j] = previous; PosmV[j] = Press*NA*kB*T*1e21; j++; Press = 0; DeltaVEnd=false; zInit = par[0];}
  i++; previous = par[0]; previousRho = par[5]+par[6];
} Plength = j; FormerPhi = fabs(par[8]);

fclose(fp);
  printf("Reading done;\n");

//  Posm = (double*) realloc (Posm, (Plength+1)*sizeof(double));
//  PosmV = (double*) realloc (PosmV, (Plength+1)*sizeof(double));
//  Width = (double*) realloc (Width, (Plength+1)*sizeof(double));

  while (i<Plength) {
    cout << setw(15) << Width[i] << setw(15) << Posm[i] << setw(15) << PosmV[i] << setw(15) << Posm[i] + PosmV[i] << endl;
  }

  strcpy(filename,"Data/"); strcat(filename, project); strcat(filename,"_DisjoiningPressure.dat");

  datafile.open(filename,ios::trunc);
  datafile << "#" << setw(49) << "1) width (nm)" << setw(50) << "2) Pressure (*0.25 kT/nm^3 or pN/nm^2)" << setw(50) << "3) Pressure due to the wall potential (*0.25 kT/nm^3 or pN/nm^2)" << setw(50) << "4) Total Pressure (*0.25 kT/nm^3 or pN/nm^2)" << endl;
  for (i=0;i<Plength;i++) {
    datafile << setw(50) << Width[i] << setw(50) << Posm[i] << setw(50) << PosmV[i] << setw(50) << Posm[i] + PosmV[i] << endl;
  }
  datafile.close();

  free(Posm); free(Width);

}



void wait_for_key () {
#if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) || defined(__TOS_WIN__)  // every keypress registered, also arrow keys
    cout << endl << "Press any key to continue..." << endl;

    FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));
    _getch();
#elif defined(unix) || defined(__unix) || defined(__unix__) || defined(__APPLE__)
    cout << endl << "Press ENTER to continue..." << endl;

    std::cin.clear();
    std::cin.ignore(std::cin.rdbuf()->in_avail());
    std::cin.get();
#endif
    return;
}


int main(){

  CoulombFluid Dielectric;
  char project[70];


  strcpy(project,"RegularizedPotential/Project_VReg_001_21_808080_900mM_N50");

  Dielectric.SetName(project);
//  Dielectric.Iterate2WallSession();
```

```
//  Dielectric.Iterate2Walls1Dielectric();

  Dielectric.Iterate2DielectricWalls();


//  Dielectric.TestConvolution();
//  Dielectric.TestGaussianConvolution();

  //Dielectric.Iterate2DielectricWallSession();


//  Dielectric.ObtainDisjoiningPressure();

//  Dielectric.test_dht_potentials();
//  test_dht_exp1();
//  Dielectric.TestReservoir();
//  Dielectric.TestMatrixInversion();

  return 0;
}
```

README

/* MIXING

*

* For Irreversible Polymerization

* Author: Prateek K. Jha (prateekjha.iitr@gmail.com)

* Current Version: 14 (Polymer Mixing)

* Revision Date: Feb 14, 2012

*

* Project Files:

* main.cpp

* system.cpp

* utils.cpp

* mc.cpp

* utils.h

* xyz.h

* energies.h

* grid.h

* header.h

* Makefile

* mc.h

* particle.h

* system.h

* chain.h

*

* Dependencies: GSL, BOOST Libraries

*

* To Run:

* make followed by ./MIXING

* ./MIXING -h for options

*/

main.cpp

```
//MIXING: main.cpp Main Program (Revision Date: Nov 1, 2011)
#include "header.h"
#include "system.h"
#include "mc.h"

int main(int argc, char *argv[])
{
        time_t start, end;
        time(&start);

    System sys;
    sys.ReadInput(argc,argv);
    sys.Create();
    sys.WriteMol2(0);
        //sys.WriteDump(0);
        //sys.WritePOV(0);
        MC mc;
        mc.S=sys;
        mc.Sweep();


        //Finalize Random number
        gsl_rng_free(sys.gsl_r);
        time(&end);
        cout<<"Time Elapsed="<<difftime(end,start)<<endl;
  return 0;
}
```

system.cpp

```
/MIXING: system.cpp System Class Function Definitions (Revision Date: Nov 1, 2011)
#include "system.h"
void System::Create()
{
   cout<<"Creating System"<<endl;
   XYZ xyz, dxyz;
   bool flag;
   double theta, phi;
   int count=0;
   //for all chains
   for(int i=0; i<NCHAINS; i++)
   {
 //    cout<<"Creating chain #:"<<i<<endl;
     Chain ch;
     //for all beads of that chains
     //Find a random initial point in (-0.5*L, 0.5*L)
     //Check for overlaps and reject if overlapping
     flag=true;
     while(flag)
     {
          xyz.set(gsl_rng_uniform(gsl_r)*L-0.5*L,gsl_rng_uniform(gsl_r)*L-
0.5*L,gsl_rng_uniform(gsl_r)*L-0.5*L);
          flag=false;
          for(int k=1; k<P.size(); k++)
          {
            if(xyz.real_d2(P[k-1].coord)<=1.0)
            {
              flag=true;
              break;
            }
```

```
        }
}

for(int j=0; j<NBEADS; j++)
{
        Particle p;
        //Check for overlaps and reject if overlapping
        flag=true;
        while(flag)
        {
         //Make a bond of length sigma in a random direction
         theta=gsl_rng_uniform(gsl_r)*2.0*PI;
         phi=gsl_rng_uniform(gsl_r)*PI;
         dxyz.set(cos(theta)*sin(phi),sin(theta)*sin(phi),cos(phi));
         dxyz=xyz+dxyz;
         flag=false;
         for(int k=1; k<P.size(); k++)
         {
          if(dxyz.real_d2(P[k-1].coord)<=1.0)
          {
           flag=true;
           break;
          }
         }
        }
        //accept
        xyz=dxyz;
        p.coord=xyz;
        p.chain_ID=i;
        p.nbonds=0;
```

```
            //assign type --Ends are B, Others are A
            if(j==0 || j==NBEADS-1)
              p.type='B';
            else
              p.type='A';
            if(j>0)
            {
              p.bonded_nbr.push_back(i*NBEADS+j-1);
              p.nbonds++;
            }
            if(j<NBEADS-1)
            {
              p.bonded_nbr.push_back(i*NBEADS+j+1);
              p.nbonds++;
            }
            NBONDS+=p.nbonds;
            P.push_back(p);
            ch.particles.push_back(i*NBEADS+j);
        }
        C.push_back(ch);
      }
      NBONDS/=2;

}

void System::WriteMol2(int timestep)
{
  //   cout<<"Writing Mol2 file"<<endl;
       ofstream out;
       char FileName[100];
```

```
    sprintf(FileName,"%s_Config_s_%.08d.mol2",MCDirName.c_str(),timestep);


    out.open(FileName);
  //    out<<"# Mol2 formatted input file for\t"<<NCHAINS<<"\tchains &\t"<<NBEADS<<"\tbeads per chain"<<endl;


    double dist2_max=0.25*L*L;
    int nbonds_display=NBONDS;
    XYZ im, im2;


    //Count the number of bonds which do not cross the boundaries
    for(int i=0; i<NT; i++)
    {
          im=image(P[i].coord,L);
          for(int j=0; j<P[i].nbonds; j++)
          {
           if(i>P[i].bonded_nbr[j])
           {
              im2=image(P[P[i].bonded_nbr[j]].coord,L);
              if(im.real_d2(im2)>dist2_max)
                    nbonds_display--;
           }
          }
    }


    out<<"@<TRIPOS>MOLECULE"<<endl;
    out<<"Polymer"<<endl;
    out<<NT<<"\t"<<nbonds_display<<endl;
    out<<"SMALL"<<endl;
    out<<"NO_CHARGES"<<endl;
```

```
    out<<"@<TRIPOS>ATOM"<<endl;


    string name,type;
    name="C.3";
    for(int i=0; i<NT; i++)
    {
            im=image(P[i].coord,L);

out<<setw(6)<<i+1<<"\t"<<P[i].type<<"\t"<<setw(8)<<im.x<<"\t"<<setw(8)<<im.y<<"\t"<<setw(8)<<i
m.z<<"\t"<<name<<endl;
    }


    out<<"@<TRIPOS>BOND"<<endl;
    int count=0;
    for(int i=0; i<NT; i++)
    {
          im=image(P[i].coord,L);
          for(int j=0; j<P[i].nbonds; j++)
          {
           if(i>P[i].bonded_nbr[j])
           {
             im2=image(P[P[i].bonded_nbr[j]].coord,L);
             if(im.real_d2(im2)<=dist2_max)

out<<setw(8)<<++count<<"\t"<<setw(8)<<i+1<<"\t"<<setw(8)<<P[i].bonded_nbr[j]+1<<"\t"<<setw(2
)<<"1"<<endl;
           }
          }
    }



    out.close();
```

```
  //   cout<<"Mol2 input written in\t"<<FileName<<endl;
    return;
}

void System::WriteDump(int timestep)
{
    char FileName[200];
    sprintf(FileName,"%s_Dump.lammpstrj",MCDirName.c_str());
    ofstream out;
    out.open(FileName,ios::app);

    out<<"ITEM: TIMESTEP"<<endl;
    out<<timestep<<endl;
    out<<"ITEM: NUMBER OF ATOMS"<<endl;
    out<<NT<<endl;
    out<<"ITEM: BOX BOUNDS"<<endl;
    out<<-0.5*L<<"\t"<<0.5*L<<endl;
    out<<-0.5*L<<"\t"<<0.5*L<<endl;
    out<<-0.5*L<<"\t"<<0.5*L<<endl;
    out<<"ITEM: ATOMS index type x y z"<<endl;
    XYZ im;
    for(int i=0; i<NT; i++)
    {
            im=image(P[i].coord,L);

out<<setw(6)<<i+1<<"\t"<<P[i].type<<"\t"<<setw(8)<<im.x<<"\t"<<setw(8)<<im.y<<"\t"<<setw(8)<<i
m.z<<endl;
    }
    out.close();
}
```

```
void System::WritePOV(int timestep)
{
    cout<<"Writing POV file"<<endl;
    ofstream out;
    char FileName[100];
    sprintf(FileName,"%s_Config_s_%.08d.pov",MCDirName.c_str(),timestep);

    out.open(FileName);
    out<<"// POV 3.x input script "<<endl;
    out<<"// try povray +W669 +H649 -Iplot.pov -Oplot.pov.tga +P +X +A +FT +C"<<endl;
    out<<"#if (version < 3.5)"<<endl;
    out<<"#error \"POV3DisplayDevice has been compiled for POV-Ray 3.5 or above.\\nPlease upgrade POV-Ray or recompile VMD.\""<<endl;
    out<<"#end"<<endl;
    out<<"#declare VMD_clip_on=array[3] {0, 0, 0};"<<endl;
    out<<"#declare VMD_clip=array[3];"<<endl;
    out<<"#declare VMD_scaledclip=array[3];"<<endl;
    out<<"#declare VMD_line_width=0.0020;"<<endl;

    out<<"#macro VMDC ( C1 )"<<endl;
    out<<"\ttexture { pigment { rgbt C1 }}"<<endl;
    out<<"#end"<<endl;

    out<<"#macro VMD_point (P1, R1, C1)"<<endl;
    out<<"\t#local T = texture { finish { ambient 1.0 diffuse 0.0 phong 0.0 specular 0.0 } pigment { C1 } }"<<endl;
    out<<"\t#if(VMD_clip_on[2])"<<endl;
    out<<"\tintersection {"<<endl;
    out<<"\t\tsphere {P1, R1 texture {T} #if(VMD_clip_on[1]) clipped_by {VMD_clip[1]} #end no_shadow}"<<endl;
    out<<"\t\tVMD_clip[2]"<<endl;
    out<<"\t}"<<endl;
```

```
    out<<"\t#else"<<endl;

    out<<"\tsphere {P1, R1 texture {T} #if(VMD_clip_on[1]) clipped_by {VMD_clip[1]} #end no_shadow}"<<endl;

    out<<"\t#end"<<endl;

    out<<"#end"<<endl;


    out<<"#macro VMD_line (P1, P2, C1)"<<endl;

    out<<"\t#local T = texture { finish { ambient 1.0 diffuse 0.0 phong 0.0 specular 0.0 } pigment { C1 } }"<<endl;

    out<<"\t#if(VMD_clip_on[2])"<<endl;

    out<<"\tintersection {"<<endl;

    out<<"\t\tcylinder {P1, P2, VMD_line_width texture {T} #if(VMD_clip_on[1]) clipped_by {VMD_clip[1]} #end no_shadow}"<<endl;

    out<<"\t\tVMD_clip[2]"<<endl;

    out<<"\t}"<<endl;

    out<<"\t#else"<<endl;

    out<<"\tcylinder {P1, P2, VMD_line_width texture {T} #if(VMD_clip_on[1]) clipped_by {VMD_clip[1]} #end no_shadow}"<<endl;

    out<<"\t#end"<<endl;

    out<<"#end"<<endl;


    out<<"#macro VMD_sphere (P1, R1, C1)"<<endl;

    out<<"\t#local T = texture { pigment { C1 } }"<<endl;

    out<<"\t#if(VMD_clip_on[2])"<<endl;

    out<<"\tintersection {"<<endl;

    out<<"\t\tsphere {P1, R1 texture {T} #if(VMD_clip_on[1]) clipped_by {VMD_clip[1]} #end no_shadow}"<<endl;

    out<<"\t\tVMD_clip[2]"<<endl;

    out<<"\t}"<<endl;

    out<<"\t#else"<<endl;

    out<<"\tsphere {P1, R1 texture {T} #if(VMD_clip_on[1]) clipped_by {VMD_clip[1]} #end no_shadow}"<<endl;

    out<<"\t#end"<<endl;
```

```
    out<<"#end"<<endl;


    out<<"#macro VMD_cylinder (P1, P2, R1, C1, O1)"<<endl;

    out<<"\t#local T = texture { pigment { C1 } }"<<endl;

    out<<"\t#if(VMD_clip_on[2])"<<endl;

    out<<"\tintersection {"<<endl;

    out<<"\t\tcylinder {P1, P2, R1 #if(O1) open #end texture {T} #if(VMD_clip_on[1]) clipped_by {VMD_clip[1]} #end no_shadow}"<<endl;

    out<<"\t\tVMD_clip[2]"<<endl;

    out<<"\t}"<<endl;

    out<<"\t#else"<<endl;

    out<<"\tcylinder {P1, P2, R1 #if(O1) open #end texture {T} #if(VMD_clip_on[1]) clipped_by {VMD_clip[1]} #end no_shadow}"<<endl;

    out<<"\t#end"<<endl;

    out<<"#end"<<endl;


    out<<"#macro VMD_cone (P1, P2, R1, C1)"<<endl;

    out<<"\t#local T = texture { pigment { C1 } }"<<endl;

    out<<"\t#if(VMD_clip_on[2])"<<endl;

    out<<"\tintersection {"<<endl;

    out<<"\t\tcone {P1, R1, P2, VMD_line_width texture {T} #if(VMD_clip_on[1]) clipped_by {VMD_clip[1]} #end no_shadow}"<<endl;

    out<<"\t\tVMD_clip[2]"<<endl;

    out<<"\t}"<<endl;

    out<<"\t#else"<<endl;

    out<<"\tcone {P1, R1, P2, VMD_line_width texture {T} #if(VMD_clip_on[1]) clipped_by {VMD_clip[1]} #end no_shadow}"<<endl;

    out<<"\t#end"<<endl;

    out<<"#end"<<endl;


    out<<"#macro VMD_triangle (P1, P2, P3, N1, N2, N3, C1)"<<endl;

    out<<"\t#local T = texture { pigment { C1 } }"<<endl;
```

```
    out<<"\tsmooth_triangle {P1, N1, P2, N2, P3, N3 texture {T} #if(VMD_clip_on[1]) clipped_by {VMD_clip[1]} #end no_shadow}"<<endl;

    out<<"#end"<<endl;


    out<<"#macro VMD_tricolor (P1, P2, P3, N1, N2, N3, C1, C2, C3)"<<endl;

    out<<"\t#local NX = P2-P1;"<<endl;

    out<<"\t#local NY = P3-P1;"<<endl;

    out<<"\t#local NZ = vcross(NX, NY);"<<endl;

    out<<"\t#local T = texture { pigment {"<<endl;

    out<<"\t\taverage pigment_map {"<<endl;

    out<<"\t\t\t[1 gradient x color_map {[0 rgb 0] [1 C2*3]}]"<<endl;

    out<<"\t\t\t[1 gradient y color_map {[0 rgb 0] [1 C3*3]}]"<<endl;

    out<<"\t\t\t[1 gradient z color_map {[0 rgb 0] [1 C1*3]}]"<<endl;

    out<<"\t\t}"<<endl;

    out<<"\t\tmatrix <1.01,0,1,0,1.01,1,0,0,1,-.002,-.002,-1>"<<endl;

    out<<"\t\tmatrix <NX.x,NX.y,NX.z,NY.x,NY.y,NY.z,NZ.x,NZ.y,NZ.z,P1.x,P1.y,P1.z>"<<endl;

    out<<"\t} }"<<endl;

    out<<"\tsmooth_triangle {P1, N1, P2, N2, P3, N3 texture {T} #if(VMD_clip_on[1]) clipped_by {VMD_clip[1]} #end no_shadow}"<<endl;

    out<<"#end"<<endl;


    out<<"camera {"<<endl;

    out<<"\torthographic"<<endl;

    out<<"\tlocation <"<<-0.6*L<<","<<-0.55*L<<","<<-0.5*L<<">"<<endl;

    out<<"\tlook_at <"<<0.45*L<<","<<0.4*L<<","<<0.5*L<<">"<<endl;

   //  out<<"\tup <"<<-0.5*L<<","<<0.5*L<<","<<-0.5*L<<">"<<endl;

   //  out<<"\tright <"<<0.5*L<<","<<-0.5*L<<","<<-0.5*L<<">"<<endl;

    out<<"}"<<endl;


    out<<"light_source {"<<endl;

    out<<"\t<"<<-0.1/5.0*L<<", "<<0.1/5.0*L<<", "<<-1.0/5.0*L<<">"<<endl;

    out<<"\tcolor rgb<1.000, 1.000, 1.000>"<<endl;
```

```
 out<<"\tparallel"<<endl;
 out<<"\tpoint_at <0.0, 0.0, 0.0>"<<endl;
 out<<"}"<<endl;

 out<<"light_source {"<<endl;
 out<<"\t<"<<1.0/5.0*L<<", "<<2.0/5.0*L<<", "<<-0.5/5.0*L<<">"<<endl;
 out<<"\tcolor rgb<1.000, 1.000, 1.000>"<<endl;
 out<<"\tparallel"<<endl;
 out<<"\tpoint_at <0.0, 0.0, 0.0>"<<endl;
 out<<"}"<<endl;

 out<<"background {"<<endl;
 out<<"\tcolor rgb<0.500, 0.300, 0.000>"<<endl;
 out<<"}"<<endl;

 out<<"#default { texture {"<<endl;
 out<<"\tfinish { ambient 0.000 diffuse 0.650 phong 0.1 phong_size 40.000 specular 0.500 }"<<endl;
 out<<"} }"<<endl;

//  out<<"#declare VMD_line_width=0.0020;"<<endl;
 out<<"// MoleculeID: 0 ReprID: 0 Beginning VDW"<<endl;
 out<<"#declare VMD_line_width=0.10;"<<endl;
 XYZ im;
 XYZ im2;
 for(int i=0; i<NT; i++)
 {
        im=image(P[i].coord,L);
        if(P[i].type=='A')
          out<<"VMD_sphere(<"<<im.x<<","<<im.y<<","<<im.z<<">,0.2,rgbt<0.0,1.0,0.0>)"<<endl;
        else
          out<<"VMD_sphere(<"<<im.x<<","<<im.y<<","<<im.z<<">,0.5,rgbt<1.0,1.0,1.0>)"<<endl;
```

```
            for(int j=0; j<P[i].nbonds; j++)
            {
              if(i>P[i].bonded_nbr[j])
              {
                im2=image(P[P[i].bonded_nbr[j]].coord,L);
                if(im.real_d(im2)<0.5*L)

           //out<<"VMD_cylinder(<"<<im.x<<","<<im.y<<","<<im.z<<">,<"<<im2.x<<","<<im2.y<<","<<i
m2.z<<">0.2,rgbt<0.500,0.500,0.500,0.000>,1)"<<endl;


out<<"VMD_line(<"<<im.x<<","<<im.y<<","<<im.z<<">,<"<<im2.x<<","<<im2.y<<","<<im2.z<<">,rgbt
<0.0,0.0,0.0,0.000>)"<<endl;
                else //break the line in two parts
                {
                      XYZ imd=im2-im;
                      XYZ pt; //new point
                      if(fabs(imd.x)>0.5*L)
                      {
                        if(fabs(imd.y>0.5*L) || fabs(imd.z>0.5*L))
                          break;
                        pt.x=0.5*L;
                        pt.y=(0.5*L-im.x)/imd.x*imd.y+im.y;
                        pt.z=(0.5*L-im.x)/imd.x*imd.z+im.z;
                        if(imd.x>0.0)
                        //
out<<"VMD_cylinder(<"<<im2.x<<","<<im2.y<<","<<im2.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">0.
2,rgbt<0.500,0.500,0.500,0.000>,1)"<<endl;


out<<"VMD_line(<"<<im2.x<<","<<im2.y<<","<<im2.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">,rgbt<
0.0,0.0,0.0,0.000>)"<<endl;
                        else
                       //
out<<"VMD_cylinder(<"<<im.x<<","<<im.y<<","<<im.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">0.2,rg
bt<0.500,0.500,0.500,0.000>,1)"<<endl;
```

```
out<<"VMD_line(<"<<im.x<<","<<im.y<<","<<im.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">,rgbt<0.0,
0.0,0.0,0.000>)"<<endl;

                    pt.x=-0.5*L;

                    pt.y=(-0.5*L-im.x)/imd.x*imd.y+im.y;

                    pt.z=(-0.5*L-im.x)/imd.x*imd.z+im.z;

                    if(imd.x<0.0)

                    //
out<<"VMD_cylinder(<"<<im2.x<<","<<im2.y<<","<<im2.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">0.
2,rgbt<0.500,0.500,0.500,0.000>,1)"<<endl;


out<<"VMD_line(<"<<im2.x<<","<<im2.y<<","<<im2.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">,rgbt<
0.0,0.0,0.0,0.000>)"<<endl;

                    else

                    //
out<<"VMD_cylinder(<"<<im.x<<","<<im.y<<","<<im.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">0.2,rg
bt<0.500,0.500,0.500,0.000>,1)"<<endl;


out<<"VMD_line(<"<<im.x<<","<<im.y<<","<<im.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">,rgbt<0.0,
0.0,0.0,0.000>)"<<endl;

                  }

                  else if(fabs(imd.y)>0.5*L)

                  {

                    if(fabs(imd.z>0.5*L) || fabs(imd.x>0.5*L))

                      break;

                    pt.y=0.5*L;

                    pt.z=(0.5*L-im.y)/imd.y*imd.z+im.z;

                    pt.x=(0.5*L-im.y)/imd.y*imd.x+im.x;

                    if(imd.y>0.0)

                    //
out<<"VMD_cylinder(<"<<im2.x<<","<<im2.y<<","<<im2.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">0.
2,rgbt<0.500,0.500,0.500,0.000>,1)"<<endl;


out<<"VMD_line(<"<<im2.x<<","<<im2.y<<","<<im2.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">,rgbt<
0.0,0.0,0.0,0.000>)"<<endl;

                    else
```

```
                    //
out<<"VMD_cylinder(<"<<im.x<<","<<im.y<<","<<im.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">0.2,rgbt<0.500,0.500,0.500,0.000>,1)"<<endl;


out<<"VMD_line(<"<<im.x<<","<<im.y<<","<<im.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">,rgbt<0.0,0.0,0.0,0.000>)"<<endl;

                    pt.y=-0.5*L;

                    pt.z=(-0.5*L-im.y)/imd.y*imd.z+im.z;

                    pt.x=(-0.5*L-im.y)/imd.y*imd.x+im.x;

                    if(imd.y<0.0)

                     //
out<<"VMD_cylinder(<"<<im2.x<<","<<im2.y<<","<<im2.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">0.2,rgbt<0.500,0.500,0.500,0.000>,1)"<<endl;


out<<"VMD_line(<"<<im2.x<<","<<im2.y<<","<<im2.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">,rgbt<0.0,0.0,0.0,0.000>)"<<endl;

                    else

                     //
out<<"VMD_cylinder(<"<<im.x<<","<<im.y<<","<<im.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">0.2,rgbt<0.500,0.500,0.500,0.000>,1)"<<endl;


out<<"VMD_line(<"<<im.x<<","<<im.y<<","<<im.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">,rgbt<0.0,0.0,0.0,0.000>)"<<endl;

                   }

                   else

                   {

                    if(fabs(imd.x>0.5*L) || fabs(imd.y>0.5*L))

                      break;

                    pt.z=0.5*L;

                    pt.x=(0.5*L-im.z)/imd.z*imd.x+im.x;

                    pt.y=(0.5*L-im.z)/imd.z*imd.y+im.y;

                    if(imd.z>0.0)

                   //
out<<"VMD_cylinder(<"<<im2.x<<","<<im2.y<<","<<im2.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">0.2,rgbt<0.500,0.500,0.500,0.000>,1)"<<endl;
```

```
out<<"VMD_line(<"<<im2.x<<","<<im2.y<<","<<im2.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">,rgbt<
0.0,0.0,0.0,0.000>)"<<endl;

                    else

                   //
out<<"VMD_cylinder(<"<<im.x<<","<<im.y<<","<<im.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">0.2,rg
bt<0.500,0.500,0.500,0.000>,1)"<<endl;


out<<"VMD_line(<"<<im.x<<","<<im.y<<","<<im.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">,rgbt<0.0,
0.0,0.0,0.000>)"<<endl;

                    pt.z=-0.5*L;

                    pt.x=(-0.5*L-im.z)/imd.z*imd.x+im.x;

                    pt.y=(-0.5*L-im.z)/imd.z*imd.y+im.y;

                    if(imd.z<0.0)

                     //
out<<"VMD_cylinder(<"<<im2.x<<","<<im2.y<<","<<im2.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">0.
2,rgbt<0.500,0.500,0.500,0.000>,1)"<<endl;


out<<"VMD_line(<"<<im2.x<<","<<im2.y<<","<<im2.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">,rgbt<
0.0,0.0,0.0,0.000>)"<<endl;

                    else

                    //
out<<"VMD_cylinder(<"<<im.x<<","<<im.y<<","<<im.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">0.2,rg
bt<0.500,0.500,0.500,0.000>,1)"<<endl;


out<<"VMD_line(<"<<im.x<<","<<im.y<<","<<im.z<<">,<"<<pt.x<<","<<pt.y<<","<<pt.z<<">,rgbt<0.0,
0.0,0.0,0.000>)"<<endl;

                  }

              }

             }

            }

    }

    out<<"#declare VMD_line_width=0.050;"<<endl;

    out<<"VMD_line(<"<<-0.5*L<<","<<-0.5*L<<","<<-0.5*L<<">,<"<<0.5*L<<","<<-0.5*L<<","<<-
0.5*L<<">,rgbt<0.250,0.750,0.750,0.000>)"<<endl;

    out<<"VMD_line(<"<<-0.5*L<<","<<-0.5*L<<","<<-0.5*L<<">,<"<<-0.5*L<<","<<0.5*L<<","<<-
0.5*L<<">,rgbt<0.250,0.750,0.750,0.000>)"<<endl;
```

```
    out<<"VMD_line(<"<<-0.5*L<<","<<-0.5*L<<","<<-0.5*L<<">,<"<<-0.5*L<<","<<-
0.5*L<<","<<0.5*L<<">,rgbt<0.250,0.750,0.750,0.000>)"<<endl;


    out<<"VMD_line(<"<<0.5*L<<","<<0.5*L<<","<<0.5*L<<">,<"<<-
0.5*L<<","<<0.5*L<<","<<0.5*L<<">,rgbt<0.250,0.750,0.750,0.000>)"<<endl;

    out<<"VMD_line(<"<<0.5*L<<","<<0.5*L<<","<<0.5*L<<">,<"<<0.5*L<<","<<-
0.5*L<<","<<0.5*L<<">,rgbt<0.250,0.750,0.750,0.000>)"<<endl;

    out<<"VMD_line(<"<<0.5*L<<","<<0.5*L<<","<<0.5*L<<">,<"<<0.5*L<<","<<0.5*L<<","<<-
0.5*L<<">,rgbt<0.250,0.750,0.750,0.000>)"<<endl;


    out<<"VMD_line(<"<<0.5*L<<","<<-0.5*L<<","<<-0.5*L<<">,<"<<0.5*L<<","<<0.5*L<<","<<-
0.5*L<<">,rgbt<0.250,0.750,0.750,0.000>)"<<endl;

    out<<"VMD_line(<"<<0.5*L<<","<<-0.5*L<<","<<-0.5*L<<">,<"<<0.5*L<<","<<-
0.5*L<<","<<0.5*L<<">,rgbt<0.250,0.750,0.750,0.000>)"<<endl;


    out<<"VMD_line(<"<<-0.5*L<<","<<0.5*L<<","<<-0.5*L<<">,<"<<0.5*L<<","<<0.5*L<<","<<-
0.5*L<<">,rgbt<0.250,0.750,0.750,0.000>)"<<endl;

    out<<"VMD_line(<"<<-0.5*L<<","<<0.5*L<<","<<-0.5*L<<">,<"<<-
0.5*L<<","<<0.5*L<<","<<0.5*L<<">,rgbt<0.250,0.750,0.750,0.000>)"<<endl;


    out<<"VMD_line(<"<<-0.5*L<<","<<-0.5*L<<","<<0.5*L<<">,<"<<0.5*L<<","<<-
0.5*L<<","<<0.5*L<<">,rgbt<0.250,0.750,0.750,0.000>)"<<endl;

    out<<"VMD_line(<"<<-0.5*L<<","<<-0.5*L<<","<<0.5*L<<">,<"<<-
0.5*L<<","<<0.5*L<<","<<0.5*L<<">,rgbt<0.250,0.750,0.750,0.000>)"<<endl;

    out<<"// End of POV-Ray 3.x generation"<<endl;

    return;

}
```

utils.cpp

```
//MIXING: utils.cpp Utilities Functions (Revision Date: Nov 1, 2011)
#include "utils.h"
//assumes particle is between -L to +L
XYZ image(XYZ p, double L)
{
  XYZ xyz;
  xyz.x=myfmod(p.x+0.5*L, L)-0.5*L;
  xyz.y=myfmod(p.y+0.5*L, L)-0.5*L;
  xyz.z=myfmod(p.z+0.5*L, L)-0.5*L;
  return xyz;
}
double myfmod(double x, double y)
{
  double temp=fmod(x,y);
  if(temp>=0.0)
    return temp;
  else
    return temp+y;
}
//Minimum image distance squared: pass original coordinates
double min_d2(XYZ a, XYZ b, double L)
{
    a=image(a,L);
    b=image(b,L);
    XYZ d=a-b;
    d.my_abs();
    if(d.x>=0.5*L)
      d.x=L-d.x;

    if(d.y>=0.5*L)
```

```
    d.y=L-d.y;

  if(d.z>=0.5*L)
    d.z=L-d.z;

  return d.norm2();
}

//Use the prescription given in http://mathworld.wolfram.com/SpherePointPicking.html
XYZ RandomMove(XYZ old, double step, double u, double v)
{
  double theta=2.0*M_PI*u;
  double phi=acos(2.0*v-1.0);
  return XYZ(old.x+step*cos(theta)*sin(phi),old.y+step*sin(theta)*sin(phi),old.z+step*cos(phi));
}

int GridIndex(XYZ p, int n, double dl, double L)
{
  int i=int(floor((p.x+0.5*L)/dl));
  int j=int(floor((p.y+0.5*L)/dl));
  int k=int(floor((p.z+0.5*L)/dl));
  return n*n*k+n*j+i;
}

int GridIndex(int i, int j, int k, int n)
{
  if(i>n-1)
    i-=n;
  else if(i<0)
    i+=n;
```

```
    if(j>n-1)
      j-=n;
    else if(j<0)
      j+=n;

    if(k>n-1)
      k-=n;
    else if(k<0)
      k+=n;

    return n*n*k+n*j+i;
}

void GridLoc(int& i , int& j, int& k, int n, int index)
{
    i=index%n;
    j=((index-i)/n)%n;
    k=(index-i-j*n)/(n*n);
}

double hfunc(double u, double Lm, double phi)
{
    return sin(2.0*PI*u/Lm+phi);
}
```

mc. cpp

```
//MIXING: mc.cpp MC Class Function Definitions (Revision Date: Feb 14,2012)
//Include Calculation of std error
#include "mc.h"
void MC::Sweep()
{
   Pn=S.P;

   double accept=0.0;

   int nsample=S.t_mixing/100;
   if(nsample<1)
     nsample=1;
   energy=TotalEnergy();
   cout<<"Initial Energy="<<energy<<endl;
   Q=S.Pe/S.L*S.MCstep;

   WriteTemplate();
         LogProfile(0,accept);
   CreateCellList();
   int move=0;

   for(int i=1; i<=S.nsweep; i++)
   {
      int nchain=S.NCHAINS;
         //stop simulation if nchain=1;
         //if(nchain==1)
           //break;

         time+=S.deltat;
         accept+=MoveParticle(move);
```

```
            //write POV if chain # is decreased
            //if(nchain!=S.NCHAINS)
            //  S.WritePOV(i);
            if(i%nsample==0)
            {
               //  S.WriteDump(i);
            //    S.WriteMol2(i);
              //   S.WritePOV(i);
                 accept/=double(nsample);
                 //This function shows false positive in valgrind. Uncomment this while running memcheck
too
                 LogProfile(i,accept);
                 accept=0.0;
            }
            if(i%S.t_mixing==0)
            {
              move++;
              move=move%3;
            }
    }
  cout<<"Final Energy="<<energy<<endl;
  cout<<"Final Energy by recalculation="<<TotalEnergy()<<endl;
    //Write Dump at the end
 //  S.WriteDump(S.nsweep);
//   S.WritePOV(S.nsweep);
  S.WriteMol2(S.nsweep);
}

void MC::WriteTemplate()
{
```

```
    ofstream out;
    string MCFileName=S.MCDirName+"_MC.log";
    out.open(MCFileName.c_str());

out<<setw(12)<<"sweep"<<"\t"<<setw(12)<<"time"<<"\t"<<setw(12)<<"accept"<<"\t"<<setw(8)<<"
Nchains"<<"\t"<<setw(12)<<"Mn"<<"\t"<<setw(12)<<"PD"<<"\t"<<setw(12)<<"<R^2>"<<"\t"<<setw
(12)<<"<A_3>"<<"\t"<<setw(12)<<"<l_b^2>"<<"\t"<<setw(12)<<"<dR^2>"<<"\t"<<setw(12)<<"<dA_
3>"<<"\t"<<setw(12)<<"<dl_b^2>"<<"\t"<<setw(12)<<"<lambda>"<<"\t"<<setw(12)<<"<dlambda>"<
<endl;
    out.close();


}


void MC::LogProfile(int i, double accept)
{
     //Calculate size measures
    int mi, ni=0, ni_mi=0, ni_mi2=0, ni_mi3=0;
    for(int j=0; j<S.C.size(); j++)
    {
                    mi=S.C[j].particles.size();
                    if(mi==0)
                      continue;
                    ni++;
                    ni_mi+=mi;
                    ni_mi2+=mi*mi;
                    ni_mi3+=mi*mi*mi;
    }

    //Include Calculation of Radius of Gyration and asphericity parameter using Rudnick and Gaspari,
Science 1987
    gsl_matrix* T = gsl_matrix_alloc(3, 3);//Gyration tensor
    gsl_vector* eval = gsl_vector_alloc(3); //Eigenvalues
```

```
    gsl_eigen_symm_workspace* w = gsl_eigen_symm_alloc(3);//Workspace for eigenvalue computation

    double mean[3];//mean coordinates

    list<int>::iterator it;


    double nav=0.0; //number of chains which are not dimers

    //we compute the averages ans std error of the averages

    double R_2=0.0, dR_2=0.0, A_3=0.0, dA_3=0.0, lb_2=0.0, dlb_2=0.0, temp=0.0, lambda=0.0, dlambda=0.0;

    //Loop over all chains

    for(int j=0; j<S.C.size(); j++)

    {

        double n=double(S.C[j].particles.size());

        //Exclude chains with <2 particles

        if(n<2.0)

          continue;

        nav=nav+1.0;

        //Compute means

        double x1=0.0, x2=0.0, x3=0.0, x1x1=0.0, x1x2=0.0, x2x2=0.0, x2x3=0.0, x3x1=0.0, x3x3=0.0, contour_length=0.0;

        double tx1, tx2, tx3;

        //iterate over particles of the chain and compute means

        for ( it=S.C[j].particles.begin() ; it != S.C[j].particles.end(); it++ )

        {

          tx1=S.P[*it].coord.x;

          tx2=S.P[*it].coord.y;

          tx3=S.P[*it].coord.z;


          x1+=tx1;

          x2+=tx2;

          x3+=tx3;

          x1x1+=tx1*tx1;
```

```
 x1x2+=tx1*tx2;
 x2x2+=tx2*tx2;
 x2x3+=tx2*tx3;
 x3x1+=tx3*tx1;
 x3x3+=tx3*tx3;

 //loop over all bonds of particle
 for(int k=0; k<S.P[*it].nbonds; k++)
  if(*it>S.P[*it].bonded_nbr[k])
  {
   temp=S.P[*it].coord.real_d2(S.P[S.P[*it].bonded_nbr[k]].coord);
   contour_length+=sqrt(temp);
   lb_2+=temp;
   dlb_2+=temp*temp;
  }
}
XYZ end(tx1,tx2,tx3);
double R_e=sqrt(S.P[*(S.C[j].particles.begin())].coord.real_d2(end));
temp=R_e/contour_length;
lambda+=temp;
dlambda+=temp*temp;
x1/=n;
x2/=n;
x3/=n;
x1x1/=n;
x1x2/=n;
x2x2/=n;
x2x3/=n;
x3x1/=n;
x3x3/=n;
//Fill the gyration tensor
```

```
            gsl_matrix_set(T,0,0,x1x1-x1*x1);
            gsl_matrix_set(T,0,1,x1x2-x1*x2);
            gsl_matrix_set(T,1,0,x1x2-x1*x2);
            gsl_matrix_set(T,1,1,x2x2-x2*x2);
            gsl_matrix_set(T,1,2,x2x3-x2*x3);
            gsl_matrix_set(T,2,1,x2x3-x2*x3);
            gsl_matrix_set(T,2,2,x3x3-x3*x3);
            gsl_matrix_set(T,0,2,x3x1-x3*x1);
            gsl_matrix_set(T,2,0,x3x1-x3*x1);

            //Compute the Eigenvalues
            gsl_eigen_symm(T, eval, w);
            //Sort the eigenvalues in ascending order
            gsl_sort_vector(eval);
            double tR1, tR2, tR3; //store temporarily eigenvalues in descending order
            tR1=gsl_vector_get(eval, 2);
            tR2=gsl_vector_get(eval, 1);
            tR3=gsl_vector_get(eval, 0);
            temp=(tR1+tR2+tR3);
            R_2+=temp;
            dR_2+=temp*temp;
            temp=0.5*((tR1-tR2)*(tR1-tR2)+(tR2-tR3)*(tR2-tR3)+(tR3-tR1)*(tR3-
tR1))/((tR1+tR2+tR3)*(tR1+tR2+tR3));
            A_3+=temp;
            dA_3+=temp*temp;
        }
        if(nav>0.0)
        {
            R_2/=nav;
            dR_2=sqrt((dR_2/nav-R_2*R_2)/nav);
            A_3/=nav;
```

```
            dA_3=sqrt((dA_3/nav-A_3*A_3)/nav);
            lb_2/=S.NBONDS;
            dlb_2=sqrt((dlb_2/S.NBONDS-lb_2*lb_2)/S.NBONDS);
            lambda/=nav;
            dlambda=sqrt((dlambda/nav-lambda*lambda)/nav);
        }
        gsl_eigen_symm_free(w);
        gsl_matrix_free(T);
        gsl_vector_free(eval);
        ofstream out;
        string MCFileName=S.MCDirName+"_MC.log";
        out.open(MCFileName.c_str(),ios::app);

out<<setw(12)<<i<<"\t"<<setw(12)<<time<<"\t"<<setw(12)<<accept<<"\t"<<setw(8)<<S.NCHAINS<<
"\t"<<setw(12)<<double(ni_mi)/double(ni)<<"\t"<<setw(12)<<double(ni_mi2*ni)/double(ni_mi*ni_
mi)<<"\t"<<setw(12)<<R_2<<"\t"<<setw(12)<<A_3<<"\t"<<setw(12)<<lb_2<<"\t"<<setw(12)<<dR_2
<<"\t"<<setw(12)<<dA_3<<"\t"<<setw(12)<<dlb_2<<"\t"<<setw(12)<<lambda<<"\t"<<setw(12)<<dla
mbda<<endl;
        out.close();
}



//Returns acceptance fraction
double MC::MoveParticle(int move)
{
   XYZ newpos;
   double rnew, rold, delta=0.0, accept=0.0;
   int i,j,k,l,new_cID, temp_cID;

   list<int>::iterator it;
   XYZ diff;
   for(k=0; k<S.NT; k++)
   {
```

```
i=gsl_rng_uniform_int(S.gsl_r,S.NT);

newpos=RandomMove(S.P[i].coord,S.MCstep,gsl_rng_uniform(S.gsl_r),gsl_rng_uniform(S.gsl_r));
delta=0.0;

    //Only iterate over the cell where the new particle is and the neighbors of that cell
    //Find id of new cell
    new_cID=GridIndex(image(newpos,S.L),n_C,d_C,S.L);
    int new_bond=-1;
    double r_newbond=S.L*S.L;
    //Add new LJ
    for(l=0; l<nbr_C; l++)
    {
      //ID of neighboring cell
      temp_cID=C[new_cID].nbr[l];
      for(it=C[temp_cID].plist.begin(); it!=C[temp_cID].plist.end(); it++)
      {

            j=*it;
            if(j!=i)
            {
              //image distance
              rnew=min_d2(S.P[j].coord,newpos,S.L);
              delta+=S.E.LJ(rnew);
              //real distance
              rnew=S.P[j].coord.real_d2(newpos);
              //also see if we need to form new bonds: Form new bonds if the particles are non-
bonded, located at the ends and less than the maximum bond length for forming a bond (real distance)
              if(S.P[j].chain_ID!=S.P[i].chain_ID && S.P[j].type=='B' && S.P[i].type=='B' &&
rnew<S.lmax_bond2)
              {
               if(rnew<r_newbond)
```

```
                {
                    r_newbond=rnew;
                    new_bond=j;
                }
               }
              }
            }
        //Subtract old LJ
        for(l=0; l<nbr_C; l++)
        {
          //ID of neighboring cell
          temp_cID=C[S.P[i].cID].nbr[l];
          for(it=C[temp_cID].plist.begin(); it!=C[temp_cID].plist.end(); it++)
          {
              j=*it;
              if(j!=i)
              {
                 rold=min_d2(S.P[j].coord,S.P[i].coord,S.L);
                 delta-=S.E.LJ(rold);
              }
          }
        }
    //add bonded FENE
    for(j=0; j<S.P[i].nbonds; j++)
    {
        rnew=newpos.real_d2(S.P[S.P[i].bonded_nbr[j]].coord);
        rold=S.P[i].coord.real_d2(S.P[S.P[i].bonded_nbr[j]].coord);
      delta+=S.E.FENE(rnew)-S.E.FENE(rold);
    }
```

```
//Add additional flow term if Pe>0
if(S.Pe>0.0)
    {
     diff=newpos-S.P[i].coord;
     if(move==0)
       delta-=Q*hfunc(S.P[i].coord.y,S.Lm,S.PHI)*diff.x/S.MCstep;
     else if(move==1)
       delta-=Q*hfunc(S.P[i].coord.z,S.Lm,S.PHI)*diff.y/S.MCstep;
     else
       delta-=Q*hfunc(S.P[i].coord.x,S.Lm,S.PHI)*diff.z/S.MCstep;
    }

if(Glauber(delta,gsl_rng_uniform(S.gsl_r)))
{
   //Accept move
   S.P[i].coord=newpos;
   accept+=1.0;
   energy+=delta;

       //make new bond if any
       if(new_bond!=-1)
       {
              int new_id=S.P[i].chain_ID;
              int old_id=S.P[new_bond].chain_ID;
              //change particle types
              S.P[i].type='A';
              S.P[new_bond].type='A';

              //change chain ID of list of the moved particles
              for ( it=S.C[old_id].particles.begin() ; it != S.C[old_id].particles.end(); it++ )
                 S.P[*it].chain_ID=new_id;
```

```
                //merge particles list of new chain from the old chain
                S.C[new_id].particles.splice(S.C[new_id].particles.end(),S.C[old_id].particles);

                //add new_bond bonded nbr for particle i and vice versa, increase #bonds by 1
                S.P[i].bonded_nbr.push_back(new_bond);
                S.P[new_bond].bonded_nbr.push_back(i);
                S.P[i].nbonds++;
                S.P[new_bond].nbonds++;

                //do not remove old chain list as it messes up old chain_IDs
                S.NCHAINS--;
                S.NBONDS++;
            }

            //update cell list if move results in change of cell
            if(new_cID!=S.P[i].cID)
            {
                C[S.P[i].cID].n--;
                C[new_cID].n++;
                C[S.P[i].cID].plist.remove(i);
                C[new_cID].plist.push_back(i);

                S.P[i].cID=new_cID;
            }
        }
    }
    return accept/double(S.NT);
}

bool MC::Glauber(double delta, double rand)
```

```
{
   if(1.0/(exp(delta)+1.0)>rand)
    return true;
   else
    return false;
}

double MC::TotalEnergy()
{
   double totalenergy=0.0;
   double r2;
   for(int i=0; i<S.NT; i++)
   {
      for(int j=0; j<S.NT; j++)
      {
            if(j!=i)
            {
         r2=min_d2(S.P[i].coord,S.P[j].coord,S.L);
              totalenergy+=S.E.LJ(r2);
            }
      }
      for(int j=0; j<S.P[i].nbonds; j++)
          {
             r2=S.P[i].coord.real_d2(S.P[S.P[i].bonded_nbr[j]].coord);
             totalenergy+=S.E.FENE(r2);
          }
   }
   return totalenergy*0.5;
}

void MC::CreateCellList()
```

```
{
   C.clear();
   nbr_C=27;
   cout<<"Creating Cell List"<<endl;
   int i,j,k;
   //Find number of cells
   n_C=int(ceil(S.L/S.lmax_bond));//cell width=maximum length to form a bond
   if(n_C<1)
     n_C=1;
   n_C3=n_C*n_C*n_C;
   //Check if number of cells> number of atoms (each cell must contain nint atoms on an average)
   int nint=1;
   if(n_C3 > nint*S.NT || n_C3<0 || n_C3!=n_C3)
   {
     cout<<" number of cells> number of atoms (adjusting s.t. each cell must contain nint atoms on an
average)"<<endl;
     n_C=int(floor(pow(double(S.NT*nint),1.0/3.0)));
     n_C3=n_C*n_C*n_C;
   }

   if(n_C<3)
   {
      cout<<"n_C<3,. Cell List will not work here.. giving up"<<endl;
      exit(1);
   }

   d_C=S.L/double(n_C);
   d_C2=d_C*d_C;

   cout<<"n_C3="<<n_C3<<endl;
```

```
//Allocate Grid
Grid g;
try
{
  C.reserve(n_C3);
}
catch (int e)
{
        cout<<"Memory issues in cell list allocation.. exiting"<<endl;
        exit(1);
}
for(k=0; k<n_C; k++)
{
   for(j=0; j<n_C; j++)
   {
     for(i=0; i<n_C; i++)
     {
        g.cm.x=(double(i)+0.5)*d_C-0.5*S.L;
        g.cm.y=(double(j)+0.5)*d_C-0.5*S.L;
        g.cm.z=(double(k)+0.5)*d_C-0.5*S.L;
        g.n=0;
             C.push_back(g);
     }
   }
}
cout<<"Filling neighbors"<<endl;
int i1, j1, k1, i2, j2, k2, indexn;
for(i=0; i<n_C3; i++)
{
  try
  {
```

```
        C[i].nbr.reserve(27);
 }
 catch(int e)
 {
         cout<<"Memory issues in nbr allocation.. exiting"<<endl;
         exit(1);
 }
}

//Fill neighbors
for(i=0; i<n_C3; i++)
{
   GridLoc(i1,j1,k1,n_C,i);
   for(i2=i1-1; i2<=i1+1; i2++)
        {
           for(j2=j1-1; j2<=j1+1; j2++)
           {
                   for(k2=k1-1; k2<=k1+1; k2++)
                   {
                      indexn=GridIndex(i2,j2,k2,n_C);
                      C[i].nbr.push_back(indexn);
                   }
           }
        }
}

cout<<"Checking list"<<endl;
for(i=0; i<n_C3; i++)
{
   if(C[i].nbr.size()!=unsigned(nbr_C))
   {
```

```
            cout<<"Error"<<"\ti="<<i<<"\tsize="<<C[i].nbr.size()<<endl;
            exit(1);
        }
    }

    cout<<"Flling particle"<<endl;
     //Fill particles
    for(i=0; i<S.NT; i++)
    {
        //Find cell for this particle
        k=GridIndex(image(S.P[i].coord,S.L),n_C,d_C,S.L);
        //Fill that cell
        C[k].plist.push_back(i);
        S.P[i].cID=k;
        C[k].n++;
    }
    cout<<"Cell List Created"<<endl;
}
```

utils.h

```
//MIXING: utils.h Utilities Heade (Revision Date: Nov 1, 2011)
#ifndef _UTILS_H
#define _UTILS_H
#include "header.h"
#include "xyz.h"
XYZ image(XYZ , double);
double min_d2(XYZ , XYZ , double );
XYZ RandomMove(XYZ , double , double , double );
int GridIndex(int , int , int , int );
int GridIndex(XYZ, int n, double w, double);
void GridLoc(int& , int& , int& , int , int );
double myfmod(double x, double y);
double hfunc(double , double, double);
#endif
```

xyz.h

```
//MIXING: XYZ Class to store and manipulate coordinates  (Revision Date: Nov 1, 2011)
#ifndef _XYZ_H
#define _XYZ_H
#include "header.h"
class XYZ
{
  public:
    double x,y,z;
    void set(double _x, double _y, double _z){x=_x;       y=_y;   z=_z;}
    XYZ(double _x, double _y, double _z){x=_x;       y=_y;   z=_z;}
    XYZ(){x=0.0;      y=0.0;            z=0.0;}
    XYZ operator + (XYZ& other){return XYZ(x+other.x,y+other.y,z+other.z);}
    XYZ operator - (XYZ& other){return XYZ(x-other.x,y-other.y,z-other.z);}
    XYZ operator / (double s){return XYZ(x/s,y/s,z/s);}
    double real_d2(XYZ& other){return (x-other.x)*(x-other.x)+(y-other.y)*(y-other.y)+(z-other.z)*(z-other.z);}
    double real_d(XYZ& other){return sqrt((x-other.x)*(x-other.x)+(y-other.y)*(y-other.y)+(z-other.z)*(z-other.z));}
    void my_abs(){x=fabs(x); y=fabs(y); z=fabs(z);}
    double norm2(){ return x*x+y*y+z*z; }
    //Image of particle between -0.5*L and 0.5*L
    void image(double L)
    {
       if(x>=0.5*L)           x-=L;
       else if(x<-0.5*L)      x+=L;

       if(y>=0.5*L)           y-=L;
       else if(y<-0.5*L)      y+=L;

       if(z>=0.5*L)           z-=L;
       else if(z<-0.5*L)      z+=L;
```

```
    }
};
#endif
```

energies.h

```
//MIXING: enegies.h Energy Class (Revision Date: Feb 2, 2012)
#ifndef _ENERGY_H
#define _ENERGY_H
#define BIGNUMBER 1.0e32
class Energy
{
public:
 //FENE Spring and Lennard Jones Hard Core
 double K, l0,l0_2, rcut, epsilon, theta, ecut;
 void Initialize(double K_FENE, double l0_FENE, double rcut_LJ, double theta_LJ)
 {
  K=K_FENE;
  l0=l0_FENE;
  l0_2=l0*l0;
  rcut=rcut_LJ;
  theta=theta_LJ;
  ecut=4.0/theta*(1.0/pow(rcut,12.0)-1.0/pow(rcut,6.0));
 }
 double FENE(double l2)
 {
    if(l2<=l0_2)
          return -0.5*K*l0_2*log(1.0-l2/l0_2)/theta;
    else
          return BIGNUMBER;
 }
 double LJ(double r2)
 {
    return 4.0/theta*(1.0/pow(r2,6.0)-1.0/(r2*r2*r2))-ecut;
 }
};
```

```
#endif
```

grid.h

```
//MIXING: grid.h Grid Class (Revision Date: September 12, 2011)
#ifndef _GRID_H
#define _GRID_H
#include "header.h"
#include "xyz.h"
class Grid
{
public:
   XYZ cm;
   int n;//Number of particles
   list<int> plist;
   vector<int> nbr; //List of neighboring cells including itself: NOT IMPLEMENTED IN GRID BUT IN CELLLIST
   Grid(){cm.x=0.0; cm.y=0.0; cm.z=0.0; n=0;}
};
#endif
```

header.h

```
//MIXING: header.h C++ and library headers (Revision Date: Jan 27, 2012)
#ifndef _HEADER_H
#define _HEADER_H
#include <fstream>
#include <iostream>
#include <iomanip>
#include <cstdio>
#include <cstdlib>
#include <cmath>
#include <vector>
#include <string>
#include <list>
#include <gsl/gsl_math.h>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_randist.h>
#include <gsl/gsl_vector.h>
#include <gsl/gsl_permutation.h>
#include <gsl/gsl_sort_vector.h>

#include <gsl/gsl_matrix.h>
#include <gsl/gsl_sort_double.h>
#include <gsl/gsl_eigen.h>

#include <boost/program_options.hpp>
#include <ctime>
#define PI 3.14159265
using namespace boost::program_options;
using namespace std;
#endif
```

Makefile

```
PROG1 = MIXING


SRC1 = main.cpp system.cpp utils.cpp mc.cpp


OBJS1 = ${SRC1:.cpp=.o}


CXX = g++ -O3

CXXFLAGS=  -lboost_program_options -lgsl -lgslcblas

LIBS=-I/software/boost/1.41.0/include/ -L/software/boost/1.41.0/lib/ -I/software/gsl/1.14/include/ -L/software/gsl/1.14/lib



#CXXFLAGS=-O3 -funroll-loops -DNDEBUG



all: $(PROG1)


$(PROG1):  $(OBJS1)

	$(CXX) $(LIBS)  $(CXXFLAGS) -o $@ $^


%.o:  %.cpp

	$(CXX) $(LIBS) -c -o $@ $<


clean:

	rm -rf *.o


distclean:

	rm -f $(PROG1) *.o *.debug
```

mc.h

```
//MIXING: mc.h MC Class (Revision Date: Jan 27, 2012)
#ifndef _MC_H
#define _MC_H
#include "system.h"
#include "utils.h"
#include "energies.h"
#include "grid.h"
#include "particle.h"
class MC
{
  public:
    System S;
    double energy, time;
   //new vector for particle positions
    vector<Particle> Pn;


    int n_C, n_C3, nbr_C; //number of nodes of cell list in one direction, and total number of cells, #neighbors of cell
    double d_C, d_C2, Q; //width of cell list and width^2


    vector<Grid> C;
    void CreateCellList();


    MC(){energy=0.0; time=0.0; }


    void WriteTemplate();
    void LogProfile(int, double );
    void Sweep();
    double MoveParticle(int);
    bool Glauber(double, double);
```

```
    double TotalEnergy();
};
#endif
```

particle.h

```
//MIXING: particle.h Particle Class (Revision Date:Nov 1, 2011)
#ifndef _PARTICLE_H
#define _PARTICLE_H
#include "header.h"
#include "xyz.h"
class Particle
{
public:
   XYZ coord;
   int chain_ID;//ID of chain
   int nbonds; //number of bonds
   int cID;//ID of cell list location
   char type;
   vector<int> bonded_nbr; //list of bonded neighbors
   Particle(){}
};
#endif
```

system.h

```
//MIXING: system.h System Class (Revision Date: Jan 6, 2012)
#ifndef _SYSTEM_H
#define _SYSTEM_H
#include "header.h"
#include "xyz.h"
#include "particle.h"
#include "chain.h"
#include "utils.h"
#include "energies.h"
class System
{
  public:
    vector<Chain> C; //List of chains
    vector<Particle> P; //List of particles
    Energy E;
    const gsl_rng_type * gsl_T;
    gsl_rng * gsl_r;
    string MCDirName;

/* INPUT PARAMETERS
 * ----------------
 * NCHAINS=Total #chains in the simulation box
 * NBEADS=#beads/chain
 * GSL_SEED=Seed for gsl random number generator
 *
 * DERIVED PARAMETERS
 * ------------------
 * NT=Total #beads=NCHAINS*NBEADS
 * L=Length of simulation box in sigma units; RHO=NT(4/3 PI)lmin^3/L^3 =>L=lmin*[NT(4/3 PI)/RHO]^(-1/3)
```

```
*/

   double L,Lm, PHI, MCstep, deltat, lmax_bond, lmax_bond2, Pe; //Length of Box, Mixing length, MC step, timestep, maximum length to form a new bond

   int NCHAINS, NBEADS, NT, GSL_SEED, NBONDS, nsweep, t_mixing; //Number of chains and number of beads/chain, and total number of beads, Seed for random number generator, total #bonds, #sweeps



   void ReadInput(int argc, char *argv[])

   {

         //inputs for Energy class

         double K_FENE, l0_FENE, rcut_LJ, theta_LJ, total_time;

      options_description desc("Usage:\nMIXING <options>");

      desc.add_options()

      ("help,h", "print usage message")

      ("Length,L", value<double>(&L)->default_value(200.0), "Box size.. (default 200.0)")

         ("Lm,M", value<double>(&Lm)->default_value(1.0), "Lm/L=(Mixing Length)/(Box size).. (default 1.0)")

         ("PHI,f", value<double>(&PHI)->default_value(0.0), "Random Phase, PHI in units of PI.. (default 0.0)")

         ("NCHAINS,C", value<int>(&NCHAINS)->default_value(100), "#chains (default 100)")

         ("NBEADS,B", value<int>(&NBEADS)->default_value(2), "#beads/chain (default 2)")

         ("K_FENE,K", value<double>(&K_FENE)->default_value(30.0), "K for FENE (default 30.0)")

         ("l0,l", value<double>(&l0_FENE)->default_value(1.5), "l0 for FENE (default 1.5)")

         ("rcut,c", value<double>(&rcut_LJ)->default_value(1.122462), "rcut for LJ(default 2^(1/6)=1.122462)")

         ("theta,t", value<double>(&theta_LJ)->default_value(1.0), "theta for LJ (default 1.0)")

         ("lmax_bond,b", value<double>(&lmax_bond)->default_value(1.5), "maximum bond length to form new bond (default 1.5)")

         ("Pe,P", value<double>(&Pe)->default_value(100.0), "Peclet Number (default 1000.0)")

         ("MCstep,m", value<double>(&MCstep)->default_value(0.10), "MC step size (default 0.1)")

         ("time,s", value<double>(&total_time)->default_value(100.0), "total time in tau_0 units (default 100.0)")
```

```
        ("GSL_SEED,g", value<int>(&GSL_SEED)->default_value(10), "seed for the RNG (default 10)");
    variables_map vm;
    store(parse_command_line(argc, argv, desc), vm);
    notify(vm);

    if (vm.count("help"))
    {
      cout << desc << "\n";
      exit(1);
    }



        Lm*=L;
        PHI*=PI;
    E.Initialize(K_FENE, l0_FENE, rcut_LJ, theta_LJ);

    //Calculate derived parameters
    NT=NCHAINS*NBEADS;
        lmax_bond2=lmax_bond*lmax_bond;
        //Initialize Random number
        gsl_rng_env_setup();


        gsl_T = gsl_rng_default;
        gsl_r = gsl_rng_alloc(gsl_T);
        gsl_rng_set(gsl_r, GSL_SEED);
        NBONDS=0;


        char fname[200];
        sprintf(fname,"_L_%.1f_NCH_%d_Pe_%.1f_T_%.2f_K_%.2f_lm_%.2f_Lm_%.2f_PHI_%.1f/",L,
NCHAINS, Pe, theta_LJ, K_FENE,lmax_bond, Lm/L, PHI/PI);
        MCDirName=fname;
```

```
        char command[500];
        sprintf(command, "mkdir %s",MCDirName.c_str());
        int k=system(command);

        deltat=1.0/12.0*MCstep*MCstep;
        nsweep=int(ceil(total_time/deltat));

        if(Pe>0.0)
        {
         t_mixing=int(Lm*Lm/(3.0*Pe*deltat));
         //t_mixing we define is one-third of that used in Vladimir's model
         cout<<"t_mixing="<<t_mixing<<endl;
         if(t_mixing<1)
         {
          cout<<"t_mixing<3... Decrease Pe"<<endl;
          exit(1);
         }
        }
        else
         t_mixing=nsweep;
   }

   void Create();
   void WriteMol2(int );
   void WriteDump(int );
   void WritePOV(int );
};
#endif
```

chain.h

```
//MIXING: chain.h Chain Class (Revision Date: Nov 1, 2011)
#ifndef _CHAIN_H
#define _CHAIN_H
#include "header.h"
class Chain
{
public:
   list<int> particles; //list of particles in the chain
};
#endif
```

KMC Magnetic Self-replication code

```
######################################################################
# This code sets up and executes the kinetic Monte Carlo
# algorithm of Prateek et al. for a system of dipolar colloids
# that form dimers as described in Dempster et al at
# http://dx.doi.org/10.1103/PhysRevE.92.042305. It will run (slowly)
# in a python interpreter but it is designed to be translated and
# compiled via shedskin. It is not optimized for python.
######################################################################

from math import sqrt, floor,pi,cos,sin,exp,ceil
from random import random
from time import clock
from os import getcwd
import sys,getopt

path=getcwd()+'/'

def zrandom(): return random()-.5

# dipole-dipole interaction energy
def E(x, y, z, R, v1x, v1y, v1z, v2x, v2y, v2z):
    return k*sig3/R**3*(v1x*v2x+v1y*v2y+v1z*v2z-3.0/R**2*\
                    (v1x*x+v1y*y+v1z*z)*(v2x*x+v2y*y+v2z*z))

# Colloid electrostatic repulsion if 1/kappa < diameter
def Q(r):
    return q*exp(-kappa*r)

# If we set all box dimensions to 1, this finds the shortest displacement vector between
#two points with periodic boundary conditions
def r(x0, y0, z0, x1, y1, z1):
    rx = x0-x1-round(x0-x1)
    ry = y0-y1-round(y0-y1)
    rz = z0-z1-round(z0-z1)
    return [rx, ry, rz, sqrt(rx**2+ry**2+rz**2)]

# same but just the distance
def just_r(x0,y0,z0, x1,y1,z1):
    rx=x0-x1-round(x0-x1)
    ry=y0-y1-round(y0-y1)
    rz=z0-z1-round(z0-z1)
    return sqrt(rx**2+ry**2+rz**2)

#rotate about an axis in lab frame
def S(axis,angle,x,y,z):
    u,v,w=axis
    assert .999999<u**2+v**2+w**2<1.000001,'non-normal rotation axis'
```

```
    c=cos(angle)
    s=sin(angle)
    return [u*(u*x+v*y+w*z)*(1-c)+x*c+(-w*y+v*z)*s,
            v*(u*x+v*y+w*z)*(1-c)+y*c+(w*x-u*z)*s,
            w*(u*x+v*y+w*z)*(1-c)+z*c+(-v*x+u*y)*s]

# external field functions
def Steady(t, M):
    return [0, 0, M]

def Mix(t, M, M0):
    #Z_PERIOD, XY_COEFF are set as globals on execution
    t0 = t*1.0/Z_PERIOD % 1.0

    if t < 20.25*Z_PERIOD: z = cos(.25/20.25*t*2*pi/Z_PERIOD)
    elif t0 < .25: z = 1 - sin(t*2*pi/Z_PERIOD)
    elif t0 < 1:  z = -1 + sin(t*2*pi/Z_PERIOD)
    elif t0 < 1.5:  z = -1 - sin(t*2*pi/Z_PERIOD)
    else:         z = 1 + sin(t*2*pi/Z_PERIOD)

    return [M*cos(XY_COEFF/Z_PERIOD*2*pi*t)*sqrt(1-z**2),
            M*sin(XY_COEFF/Z_PERIOD*2*pi*t)*sqrt(1-z**2),
            M*z]

# for normalizing vectors
def makenormal(x, y, z, normto=1.0):
    normer=normto*1.0/sqrt(x**2+y**2+z**2)
    return [x*normer,y*normer,z*normer]

# algorithm uses the box method for finding neighbors
def tobox(x, y, z):
    return int(floor((x+.5)/RC)+floor((y+.5)/RC)*nbox+floor((z+.5)/RC)*nbox**2)

# contains a list of particles and a list of neighbors
class Box:
    def __init__(self,i):
        self.parts=set([])
        neighbors=[]
        for dz in [0, -1, 1]:
            for dy in [0, -1, 1]:
                for dx in [0, -1, 1]:
                    ip = (i/(nbox**2)+dz)%nbox*nbox**2+\
                      (i/nbox+dy)%nbox*nbox+(i+dx)%nbox
                    neighbors.append(ip)
        self.neighbors=list(set(neighbors))
    def part(self,p):
        self.parts.add(p)
    def depart(self,p):
        self.parts.remove(p)
```

```
# particles
class P:
    def __init__(self, x,y,z, is_dyad=False):
        self.x, self.y, self.z = x, y, z
        self.dyad = is_dyad
        self.box = tobox(x, y, z)
        assert self.box<=nbox**3, str(self.x)+' '+str(self.y)+' '+str(self.z)+' '+str(self.box)
        boxes[self.box].part(self)
        self.vx, self.vy, self.vz = 0,0,1#dipole vector
        self.tx, self.ty, self.tz = 1,0,0#sticky point

        for i in range(3):
            axis = makenormal(zrandom(), zrandom(), zrandom())
            rotangle = 2*pi*zrandom()
            #dipole direction
            self.vx, self.vy, self.vz = S(axis, rotangle,
                                self.vx, self.vy, self.vz)
            #binding site direction
            self.tx, self.ty, self.tz = S(axis, rotangle,
                                self.vx, self.vy,self.vz)
        #locations of binding sites on colloid surfaces
        self.TX = self.x+.5*sig*self.tx
        self.TY = self.y+.5*sig*self.ty
        self.TZ = self.z+.5*sig*self.tz
        self.gx, self.gy, self.gz=x, y, z
        self.gvx, self.gvy, self.gvz = self.vx, self.vy, self.vz
        self.gtx, self.gty, self.gtz = self.tx, self.ty, self.tz
        self.normalize()
        self.gTX, self.gTY, self.gTZ = self.TX, self.TY, self.TZ
        self.deltaE = 0
        global particles
        particles.add(self)

    def normalize(self):#cleans up floating point errors from rotations
        (self.tx, self.ty, self.tz) = makenormal(self.tx, self.ty, self.tz)
        dotvt = self.vx*self.tx+self.vy*self.ty+self.vz*self.tz
        #Gram-Schmidt
        self.vx -= dotvt*self.tx
        self.vy -= dotvt*self.ty
        self.vz -= dotvt*self.tz
        (self.vx, self.vy, self.vz) = makenormal(self.vx, self.vy, self.vz)
        dotvt = self.vx*self.tx+self.vy*self.ty+self.vz*self.tz
        assert abs(dotvt)<.00001, str(dotvt)

        #class members with leading g are "ghost" particles for trying moves
        self.gtx, self.gty, self.gtz = makenormal(self.gtx, self.gty, self.gtz)
        dotvt = self.gvx*self.gtx + self.gvy*self.gty + self.gvz*self.gtz
        self.gvx -= dotvt*self.gtx
```

```
    self.gvy -= dotvt*self.gty
    self.gvz -= dotvt*self.gtz

    self.gvx, self.gvy, self.gvz = makenormal(self.gvx, self.gvy, self.gvz)
    dotvt = self.gvx*self.gtx + self.gvy*self.gty + self.gvz*self.gtz
    assert abs(dotvt)<.00001, str(dotvt)

def ghost(self): #find new coordinates for a trial move
    if random()>.75: #translate
        [ax, ay, az] = makenormal(zrandom(), zrandom(), zrandom(), a)
        self.gx = self.x+ax
        self.gy = self.y+ay
        self.gz = self.z+az
    else: #rotate
        axis=makenormal(zrandom(),zrandom(),zrandom())
        self.gvx, self.gvy, self.gvz = S(axis, angle,
                            self.vx, self.vy, self.vz)
        self.gtx, self.gty, self.gtz = S(axis, angle,
                            self.tx, self.ty, self.tz)
    self.gTX = self.gx+.5*sig*self.gtx
    self.gTY = self.gy+.5*sig*self.gty
    self.gTZ = self.gz+.5*sig*self.gtz
    assert just_r(self.x, self.y, self.z, self.gx, self.gy, self.gz)<=1.01*a, "monomer ghost error"

def forcemove(self): #move acceptance - ghost coordinates become actual coordinates
    self.x = (self.gx+.5)%1-.5
    self.y = (self.gy+.5)%1-.5
    self.z = (self.gz+.5)%1-.5
    self.vx, self.vy, self.vz = self.gvx, self.gvy, self.gvz
    self.tx, self.ty, self.tz = self.gtx,self.gty,self.gtz
    self.TX, self.TY, self.TZ = self.gTX, self.gTY, self.gTZ
    self.normalize()
    box2 = tobox(self.x, self.y, self.z)
    if box2 != self.box:
        boxes[self.box].depart(self)
        self.box = box2
        boxes[self.box].part(self)

#find the energy cost of a move
def energy(self):
    global U, particles, boxes
    dyadize = False
    U0 = 0 #stores current energy
    U1 = 0 #stores energy after the move
    legal=True
    for neighb in boxes[self.box].neighbors:
        if legal:
            if neighb == self.box: Set = boxes[neighb].parts-set([self])
            else: Set = boxes[neighb].parts
```

```
            if self.dyad:
                Set = Set - set([self.dyadpair])
            for p in Set:
                r1x, r1y, r1z, r1 = r(self.gx, self.gy, self.gz, p.x, p.y, p.z) #ghost
                if r1 > sig: #move not disallowed by hard core potential

                    r0x, r0y, r0z, r0 = r(self.x, self.y, self.z, p.x, p.y, p.z)
                    if r0 < RC:
                        U0 += E(r0x, r0y, r0z, r0, self.vx, self.vy, self.vz, p.vx, p.vy, p.vz)
                        if r0 < 2*sig: #cutoff distance for repulsion
                            U0 += Q(r0 - sig)
                    if r1 < RC:
                        U1 += E(r1x, r1y, r1z, r1, self.gvx, self.gvy, self.gvz, p.vx, p.vy, p.vz)
                        if r1 < 2*sig:
                            U1 += Q(r1 - sig)
                        if r1 < brangetest: # cutoff to check for binding event
                            rT = just_r(self.gTX, self.gTY, self.gTZ, p.TX, p.TY, p.TZ)
                            if not self.dyad and not p.dyad and rT<=brange:
                                # set variables to create dimer
                                U1 += bE
                                dyadize = True
                                self.newdyad = p

                elif r1 <= sig:
                    legal = False
                    break
        else: break
    if legal:
        U1 -= self.gvx*Mx + self.gvy*My + self.gvz*Mz
        U0 -= self.vx*Mx + self.vy*My + self.vz*Mz
    self.deltaE = U1-U0
    return (legal, dyadize)

def move(self, dyadize):
    if abs(self.deltaE)<=20: P = 1.0/(1+exp(self.deltaE)) #move acceptance
    elif self.deltaE < -20: P = 1 #avoid overflow errors
    elif self.deltaE > 20: P = 0
    prob = random()
    if P > prob:
        self.forcemove()
        global U
        U += self.deltaE
        global moved
        moved += 2
        if dyadize:
            global ndyads
            ndyads += 1
            self.dyad = True
            self.newdyad.dyad = True
```

```
            if allow_dyad:
                Dyad(self,self.newdyad) #particles now move as dimer members
        else:
            self.gx, self.gy, self.gz = self.x, self.y, self.z
            self.gvx, self.gvy, self.gvz = self.vx, self.vy, self.vz
            self.gtx, self.gty, self.gtz = self.tx, self.ty, self.tz

#pair of bound particles
class Dyad:
    def __init__(self, p1, p2):
        self.p1 = p1
        p1.dyad = True
        p2.dyad = True
        self.p2 = p2
        self.normalize()
        self.deltaE = 0
        self.p1.dyadpair = self.p2
        self.p2.dyadpair = self.p1
        self.p1.dyadmaster = self
        self.p2.dyadmaster = self
        global dyads, remove_p
        dyads.add(self)
        remove_p.append(p1)
        remove_p.append(p2)

    def axialize(self):
        self.axisx, self.axisy, self.axisz, self.axisd = r(self.p2.x, self.p2.y, self.p2.z,
                                                self.p1.x, self.p1.y, self.p1.z)
        self.gaxisx, self.gaxisy, self.gaxisz, self.axisd = r(self.p2.x, self.p2.y, self.p2.z,
                                                self.p1.x, self.p1.y, self.p1.z)
        self.axisN = makenormal(self.axisx, self.axisy, self.axisz)
        self.negaxisN = [-self.axisN[0],-self.axisN[1],-self.axisN[2]]

    def normalize(self):
        self.axialize()
        [self.p1.tx, self.p1.ty, self.p1.tz] = self.axisN
        [self.p2.tx, self.p2.ty, self.p2.tz] = self.negaxisN
        [self.p1.gtx, self.p1.gty, self.p1.gtz] = self.axisN
        [self.p2.gtx, self.p2.gty, self.p2.gtz] = self.negaxisN
        self.p1.normalize()
        self.p2.normalize()

    def ghost(self):
        global da1, da2 #diffusion coeffecticients
        coin = random()
        if coin < .375: # translate along axis of dimer
            whichway=2*round(random())-1
            ax, ay ,az = self.axisx*whichway*da1, self.axisy*whichway*da1, self.axisz*whichway*da1
            for p in self.p1, self.p2:
```

```
        p.gx, p.gy, p.gz = p.x+ax, p.y+ay, p.z+az
elif coin < .5: #translate perpendicular to axis
    [ax, ay, az]=makenormal(zrandom(),zrandom(),zrandom())
    dotaxis=ax*self.axisN[0]+ay*self.axisN[1]+az*self.axisN[2]
    ax -= dotaxis*self.axisN[0]
    ay -= dotaxis*self.axisN[1]
    az -= dotaxis*self.axisN[2]
    [ax,ay,az] = makenormal(ax, ay, az, da2)
    for p in self.p1,self.p2:
        p.gx, p.gy, p.gz = p.x+ax, p.y+ay, p.z+az
elif coin < .875: #rotate perpendicular to axis
    [ax, ay, az] = makenormal(zrandom(),zrandom(),zrandom())
    dotaxis = ax*self.axisN[0]+ay*self.axisN[1]+az*self.axisN[2]
    ax -= dotaxis*self.axisN[0]
    ay -= dotaxis*self.axisN[1]
    az -= dotaxis*self.axisN[2]
    axis = makenormal(ax,ay,az)
    (self.gaxisx, self.gaxisy, self.gaxisz) = S(axis,dangle2,self.axisx,self.axisy,self.axisz)
    (self.gaxisx, self.gaxisy, self.gaxisz) = makenormal(self.gaxisx, self.gaxisy, self.gaxisz,
                                            sig+.5*brange)
    self.gaxisd = sqrt(self.gaxisx**2+self.gaxisy**2+self.gaxisz**2)
    self.centerx = self.p1.x + .5*self.axisx
    self.centery = self.p1.y + .5*self.axisy
    self.centerz = self.p1.z + .5*self.axisz
    self.p1.gx = self.centerx - .5*self.gaxisx
    self.p1.gy = self.centery - .5*self.gaxisy
    self.p1.gz = self.centerz - .5*self.gaxisz
    self.p2.gx = self.centerx + .5*self.gaxisx
    self.p2.gy = self.centery + .5*self.gaxisy
    self.p2.gz = self.centerz + .5*self.gaxisz
    self.p1.gvx, self.p1.gvy, self.p1.gvz = S(axis, dangle2,
                        self.p1.vx, self.p1.vy, self.p1.vz)
    self.p1.gtx, self.p1.gty, self.p1.gtz = S(axis, dangle2,
                        self.p1.tx, self.p1.ty, self.p1.tz)
    self.p2.gvx, self.p2.gvy, self.p2.gvz = S(axis, dangle2,
                        self.p2.vx, self.p2.vy, self.p2.vz)
    self.p2.gtx, self.p2.gty, self.p2.gtz = S(axis, dangle2,
                        self.p2.tx, self.p2.ty, self.p2.tz)
else: #each colloid twists independently around the axis
    for p in self.p1, self.p2:
        whichway = 2*round(random())-1
        p.gvx, p.gvy, p.gvz=S(self.axisN, whichway*dangle1,
                        p.vx, p.vy, p.vz)
for p in self.p1, self.p2:
    p.gTX, p.gTY, p.gTZ=p.gx+.5*sig*p.gtx, p.gy+.5*sig*p.gty, p.gz+.5*sig*p.gtz
assert just_r(self.p1.x, self.p1.y, self.p1.z, self.p1.gx, self.p1.gy, self.p1.gz)<1.5*a, \
        'dyad ghost error %f %f %f %f %f %f' % (self.p1.x, self.p1.y, self.p1.z,
                                        self.p1.gx, self.p1.gy, self.p1.gz)
```

```
    def energy(self): #calculate energy for current and trial coordinates
        legal1, null1=self.p1.energy()
        legal2, null2=self.p2.energy()
        if legal1 and legal2:
            self.deltaE=self.p1.deltaE+self.p2.deltaE+\
                    E(self.gaxisx, self.gaxisy, self.gaxisz, self.axisd, self.p1.gvx, self.p1.gvy, self.p1.gvz,
                      self.p2.gvx, self.p2.gvy, self.p2.gvz)-\
                    E(self.axisx, self.axisy, self.axisz, self.axisd, self.p1.vx, self.p1.vy, self.p1.vz,
                      self.p2.vx, self.p2.vy, self.p2.vz)
        return (legal1 and legal2)

    def move(self): #decide whether to accept the move
        if self.deltaE < -20: P = 1
        elif self.deltaE > 20: P = 0
        else: P=1.0/(1+exp(self.deltaE))
        if P>random():
            for p in self.p1, self.p2:
                p.forcemove()
            self.normalize()
            global U
            U+=self.deltaE
            global moved
            moved+=1
        else:
            for p in self.p1, self.p2:
                p.gx, p.gy, p.gz = p.x, p.y, p.z
                p.gvx, p.gvy, p.gvz = p.vx, p.vy, p.vz
                p.gtx, p.gty, p.gtz = p.tx, p.ty, p.tz

def add_particle(is_dyad=False): #find a free position so particles are > sigma apart
    placed = False
    while not placed:
        placeable = True
        x, y, z = zrandom(), zrandom(), zrandom()
        testbox = tobox(x, y, z)
        for neighb in boxes[testbox].neighbors:
            for p in boxes[neighb].parts:
                if just_r(x, y, z, p.x, p.y, p.z) < sig:
                    placeable = False
                    break
        if placeable:
            P(x, y, z, is_dyad)
            placed = True

#create simulation with particles and a specified number of dimers
def setup(n, makedyads):
    global particles, dyads, boxes, U, ndyads
    U=0
    particles = set([])
```

```
    dyads = set([])
    boxes = []
    ndyads = 0
    for i in range(nbox**3):
        boxes.append(Box(i))
    for i in range(n):
        add_particle()
    for i in range(makedyads):
        if allow_dyad:
            for p in particles:
                x, y, z = p.x+1.01*sig*p.tx, p.y+1.01*sig*p.ty, p.z+1.01*sig*p.tz
                placeable = True
                for p2 in particles:
                    if just_r(x, y, z, p2.x, p2.y, p2.z) < sig:
                        placeable=False
                        break
                if placeable:
                    newp = P(x-round(x),y-round(y),z-round(z))
                    [newp.tx, newp.ty, newp.tz, newp.gtx, newp.gty, newp.gtz] = [
                        -p.tx,-p.ty,-p.tz]*2
                    [newp.TX, newp.gTX] = [newp.x+.5*sig*newp.tx]*2
                    [newp.TY, newp.gTY] = [newp.y+.5*sig*newp.ty]*2
                    [newp.TZ, newp.gTZ] = [newp.z+.5*sig*newp.tz]*2
                    ndyads+=1
                    dotv = newp.tx*newp.vx+newp.ty*newp.vy+newp.tz*newp.vz
                    newp.vx -= dotv*newp.vx
                    newp.vy -= dotv*newp.vy
                    newp.vz -= dotv*newp.vz
                    [newp.vx, newp.vy, newp.vz, newp.gvx, newp.gvy, newp.gvz] = makenormal(
                        newp.vx, newp.vy, newp.vz)*2
                    Dyad(p,newp)
                    break
        else:
            add_particle(is_dyad=True)


def simulate(n, makedyads, sweeptime, timeinc): #create and run a simulation for specified time
    global direction, ndyads, moved, Mx, My, Mz, remove_p, tau, tm

    f = open(path+name+'.txt','w+')
    f.write('0.0 %i \n' %makedyads)
    f.close()

    moved = 0.0
    delete_p = []
    remove_p = []
    delete_dyad = []
    setup(n, makedyads)
    Uout = []
```

```
dyadout = []
t0 = clock()
if xyz: f = open(path+name+'.xyz', 'w+')
simtime = 0.0
mixtime = 0.0
sweep = 0
mix = False
while simtime < sweeptime:
   ndyads0 = ndyads
   sweep += 1
   mixtime += timeinc
   #status of external fields - refer to Dempster et al. for explanation
   if mixtime  > tm and mix:
      print 'stopping mix at %i, number of dimers %i, time %.0f' % (
         sweep/savef, ndyads, simtime)
      mix = False
      mixtime = 0.0
   elif mixtime > tau and not mix:
      print 'starting mix at %i, number of dimers %i, time %.0f' % (
         sweep/savef, ndyads, simtime)
      mix = True
      mixtime = 0.0
   if mix: [Mx, My, Mz] = Mix(mixtime, M, M0)
   else: [Mx, My, Mz] = Steady(mixtime, M0)
   for p in particles:
      if random() > .5:
         p.ghost()
         legal1,dyadize1 = p.energy()
         if legal1: p.move(dyadize1)
   for d in dyads:
      d.ghost()
      legal1 = d.energy()
      if legal1: d.move()
   for p in remove_p:
      particles.remove(p)
   remove_p = []

   if ndyads != ndyads0:
         with open(path+name+'.txt','a') as f:
            f.write('%f %i\n' % (simtime, ndyads))


   if sweep%savef == 0:
      if xyz:
         f.write(str(3*(n+makedyads))+'\n')
         f.write('frame '+str(sweep)+'\n')
         for p in particles:
            f.write('O '+str(1000.*p.x)+' '+str(1000.*p.y)+' '+\
               str(1000.*p.z)+'\n')
```

```
                f.write('H '+str(1000.*(p.x+.5*sig*p.vx))+' '+str(1000.*(p.y+.5*sig*p.vy))+' '+\
                    str(1000.*(p.z+.5*sig*p.vz))+'\n')
                f.write('He '+str(1000.*p.TX)+' '+str(1000.*p.TY)+' '+\
                    str(1000.*p.TZ)+'\n')
            for d in dyads:
                for p in d.p1, d.p2:
                    f.write('C '+str(1000.*p.x)+' '+str(1000.*p.y)+' '+\
                        str(1000.*p.z)+'\n')
                    f.write('H '+str(1000.*(p.x+.5*sig*p.vx))+' '+str(1000.*(p.y+.5*sig*p.vy))+' '+\
                        str(1000.*(p.z+.5*sig*p.vz))+'\n')
                    f.write('He '+str(1000.*p.TX)+' '+str(1000.*p.TY)+' '+\
                        str(1000.*p.TZ)+'\n')

        simtime += timeinc

    print '100%% done at %i, %i dimers, %f move rate, %i sweeps' %(
        int(clock()-t0), ndyads, (moved*1.0/(len(particles)*sweep)), (sweep-1))
    if xyz: f.close()
    f = open(path+name+'.txt','a')
    print path+name
    f.write(str(clock()-t0)+'\n')
    f.write(str(Uout[1::])+'\n')
    f.write(str(dyadout)+'\n')
    f.write(
        'n='+str(n)+' time='+str(sweeptime)+' sig='+str(sig)+' aratio='+str(aratio)+' k='+str(k)+
        ' q='+str(q)+' M='+str(M)+' M0='+str(M0)+' tau='+str(int(tau))+' tm='+str(int(tm))+
        ' RC='+str(RC)+' bE='+str(bE)+' brangeratio='+str(brangeratio)
        )
    f.close()


if __name__=="__main__":
    allow_dyad=True
    name = 'Runs/'
    density = .001 #n*sigma**3 *pi/6
    k=20. #this is k |p|^2 / KbT*sig^3
    q = 12.0 #electrostatic repulsion
    kappa = 5.0 #electrostatic decay
    bE=-5. #binding energy - not really important for simulation
    Lm=1.0
    brangeratio=.1 #range for bonding
    aratio=.1 #move size
    tau = 300 #time between mixings
    tm = 49 #mixing time
    Z_PERIOD = 2.0 #characteristics of mixing field
    XY_COEFF = 4*sqrt(2)
    M = 200.0 #mixing field strength
    M0 = 4.0 #regular field strength
    RCratio = 5.0 #cutoff for dipole potential in sigma units
```

```
savef = 250
xyz = True #log positions
n = 990
makedyads = 10
sweeptime = 1
iteration = 0
opts, args = getopt.getopt(sys.argv[1:],"hoD:k:q:K:t:T:L:b:B:a:M:Q:Z:X:C:m:f:xn:d:N:i:")
for o, arg in opts:
    if o=='-h':
        print '-D: density, default '+str(density)
        print '-o: turn off dyad formation (to measure dyad rate without autocatalyzation)'
        print '-k: initial dipole interaction strength |p|^2 / (4*pi*sig^3*KbT), default '+str(k)
        print '-q: electrostatic repulsion between colloids, default '+str(q)
        print '-K: electrostatic decay, default '+str(kappa)
        print '-e: Binding energy, default '+str(bE)
        print '-L: length for mixing inversion, default '+str(Lm)
        print '-b: binding distance in hard core diameters, default '+str(brangeratio)
        print '-a: move distance in units of hard core diameter, default '+str(aratio)
        print '-t: time between magnetic stirring, default '+str(tau)
        print '-T: time to magnetically stir, default '+str(tm)
        print '-M: strength of mixing field, default '+str(M)
        print '-Q: strength of constant field, default '+str(M0)
        print '-Z: period of mixing field, default '+str(Z_PERIOD)
        print '-X: mixing field spin rate in the xy plane, default ' + str(XY_COEFF)
        print '-C: cut-off radius for dipole potential in hard core diameters, default '+str(RCratio)
        print '-m: name of output files, default '+name
        print '-f: save frequency, default '+str(savef)
        print '-x: turn off xyz file output'
        print '-n: number of particles, default '+str(n)
        print '-d: number of extra particles to add as dyad pairs, default '+str(makedyads)
        print '-N: total time to run simulation, default '+str(sweeptime)
        print '-i: iteration number for multiple trials, default 0'
        dummy=raw_input()
        sys.exit()
    elif o=="-o": allow_dyad=False
    elif o=="-D": density = float(arg)
    elif o=="-k": k=float(arg)
    elif o=="-q": q=float(arg)
    elif o=="-K": kappa=float(arg)
    elif o=="-e": bE=float(arg)
    elif o=="-L": Lm=float(arg)
    elif o=="-b": brangeratio=float(arg)
    elif o=="-a": aratio=float(arg)
    elif o=="-t": tau = float(arg)
    elif o=="-T": tm = float(arg)
    elif o=="-O": tstirdirection=int(arg)
    elif o=="-M": M=float(arg)
    elif o=="-Q": M0=float(arg)
    elif o=="-Z": Z_PERIOD = float(arg)
```

```
    elif o=="-X": XY_COEFF = float(arg)
    elif o=="-C": RCratio=float(arg)
    elif o=="-m": name=arg
    elif o=="-f": savef=int(arg)
    elif o=="-x": xyz=False
    elif o=="-n": n=int(arg)
    elif o=="-d": makedyads=int(arg)
    elif o=="-N": sweeptime = float(arg)
    elif o=="-i": iteration = int(arg)
name = name + '_n%i_d%i_D%.4f_k%i_q%.1f_K%.1f_b%.2f_t%i_T%i_Q%.1f_M%.1f_%i' %(
    n, makedyads, density, int(k), q, kappa, brangeratio, int(tau), int(tm),
    M0, M, iteration)
sig3 = density / (n+makedyads)*6.0/pi
sig = sig3**(1.0/3)
kappa *= 1.0/sig
brange = brangeratio*sig
RC = RCratio*sig
a = aratio*sig
angle = aratio
brangetest = brange+sig
da1 = .51*a #diffusion coefficients for dimer translation
da2 = .68*a
dangle1 = aratio*1.0/sqrt(3) #same, for dimer rotation
dangle2 = .42*aratio
nboxa = round(1.0/float(RC))
nbox = int(nboxa)
RC = 1.0/nbox
direction = 0
timeinc = aratio**2 *1.0/16
print "number of sweeps %i" %int(sweeptime/timeinc + 1)
simulate(n, makedyads, sweeptime, timeinc)
```

```
#include "utils/nr.h"
#include "PBcalculationRK.h"

#include <fftw3.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <iostream>
#include <time.h>
#include <iomanip>
#include <fstream>
using namespace std;
#include "utils/nr.h"
#include "gnuplot_i.hpp"
#include <gsl/gsl_dht.h>
#include <gsl/gsl_sf_bessel.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_rng.h>

#if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) || defined(__TOS_WIN__)
 #include <conio.h>     //for getch(), needed in wait_for_key()
 #include <windows.h> //for Sleep()
#endif


void wait_for_key(); // Program halts until keypress

#define pi 3.14159265358979323846264 3
#define NA 0.60221415
#define e0 8.854187817620e-12
#define e 1.60217646e-19
#define kB 1.3806503e-23

#define T 293.0
#define ew 80.1 /* at 293 K */ //78.4

#define ImposeLocalChargeNeutrality true

/*
------------------------------------------------------------------
-----------------------------
```

```
Calculates the radial distribution function of the primitive model (HS + Coulomb
interactions),
the equation of state, and compressibility, following several methods (HNC,MSA,PB,SCA)

   (c) by Jos W. Zwanikken, August 18, 2014
   Monica Olvera de la Cruz group
   Northwestern University
   Evanston Il


--------------------------------------------------------------------------
-----------------------------
*/


class Ion {

   public:

      double Charge, Diameter, Concentration;
      char name[26];

      Ion ();
      void PrintProperties();

};

Ion::Ion(){

   Charge = 1.0;                          /* Charge in units of elementary charge */
   Diameter = 0.6;                  /* Diameter in nanometer */
   Concentration = 0.1 * NA;        /* Bulk concentration in particles per cubic nm */
   strcpy (name,"charged sphere");  /* Name of the ion */

}

void Ion::PrintProperties() {

   cout << name << ":" << "Charge = " << Charge << " e₩tDiameter = " << Diameter << "
nm₩t Concentration = " << Concentration/NA << " M" << endl;

}
```

```
void SetIonParameters (int Ni, int *N, Ion *iontype) {

   double chargeneutral = 0, totaldensity;
   int i;
   Ion IonData[10];

   strcpy(IonData[0].name,"sodium ion");
   strcpy(IonData[1].name,"chloride ion");
   strcpy(IonData[2].name,"calcium ion");
   strcpy(IonData[3].name,"phosphate ion");
   strcpy(IonData[4].name,"neg. charged monomer");
   strcpy(IonData[5].name,"pos. charged monomer");
   strcpy(IonData[6].name,"neutral monomer");
   strcpy(IonData[7].name,"Lanthanide");

   IonData[0].Charge = 1.0;    IonData[0].Diameter = 0.6;
   IonData[1].Charge = -1.0;   IonData[1].Diameter = 0.6;
   IonData[2].Charge = 2.0;    IonData[2].Diameter = 0.6;
   IonData[3].Charge = -2.0;   IonData[3].Diameter = 0.6;
   IonData[4].Charge = -1.0;   IonData[4].Diameter = 0.6;
   IonData[5].Charge = 1.0;    IonData[5].Diameter = 0.6;
   IonData[6].Charge = 0.0;    IonData[6].Diameter = 0.6;
   IonData[7].Charge = 3.0;    IonData[7].Diameter = 0.8;

   totaldensity = 10.0 *NA;

   IonData[0].Concentration = 1.0 * totaldensity;       // * totaldensity /
fabs(IonData[0].Charge);
   IonData[1].Concentration = 1.0 * totaldensity;       // * totaldensity /
fabs(IonData[1].Charge);
   IonData[2].Concentration = 0.5 * totaldensity;       // * totaldensity /
fabs(IonData[2].Charge);
   IonData[3].Concentration = 0.5 * totaldensity;       // * totaldensity /
fabs(IonData[3].Charge);
   IonData[4].Concentration = 1.0 * totaldensity / fabs(IonData[0].Charge);
   IonData[5].Concentration = 1.0 * totaldensity / fabs(IonData[1].Charge);
   IonData[6].Concentration = 1.0 * totaldensity;
   IonData[7].Concentration = 1.0 * totaldensity;

   for (i=0;i<Ni;i++) {
      iontype[i].Charge = IonData[N[i]].Charge;
      iontype[i].Diameter = IonData[N[i]].Diameter;
      iontype[i].Concentration = IonData[N[i]].Concentration;
      strcpy(iontype[i].name,IonData[N[i]].name);
```

```
  }

  for (i=0;i<Ni;i++) chargeneutral += iontype[i].Concentration * iontype[i].Charge;
  if (chargeneutral != 0) {
    cout << "warning, the system is not charge neutral, but has an excess charge of " <<
chargeneutral << " e / nm^3" << endl;
    iontype[0].Concentration -= chargeneutral/iontype[0].Charge;
    cout << "Bulk concentration of ion type 1 (" << iontype[0].name << ") adapted to a
new value of " << iontype[0].Concentration/NA << " M" << endl;
    if(iontype[0].Concentration < 0) cout << "which is negative (!). Program does not
know how to correct.";
    chargeneutral=0;
    for (i=0;i<Ni;i++) chargeneutral += iontype[i].Concentration * iontype[i].Charge;
    cout << "total charge is now " << chargeneutral << " e / nm^3" << endl;
  }


}



class RDFevaluator {

  double kappaD,fraction, Gamma, tolerance, alpha, turn, turnstep;
  double lB,R,RD,Rmin,*r, *K, *phiZeta, *rho_res, *mu_res, phiD, *guessPhi,
*guessPhiMem;
  double *mu_exc, F_exc, Pi_exc, *N_ads, *omega_ads;
  double *hBigMem, *cBigMem, *hBigMem2, *cBigMem2, *guessPhiBigMem,
*guessPhiBigMem2;
  int i,j,k,Ni,Nres, itermax, method, evaluations;
  char projectname[70],methodname[7];
  int N;
  bool FirstRecording, HSC, FastRun;
  Mat_DP d, rho, delta, phi, roots;
  Mat3D_DP h,c,hnew,cnew,hFt,cFt,hDH,hLDH,hMem,cMem,cprev,f,u,uFT;
  Ion *ion;

  void SetPotential(bool);
  void ShiftGrid();
  double Qtot(double, double, int);
  double ShootPhiDHEMSA (int, double, int);
  void EvaluatePhiDHEMSA ();
  void EvaluateHNCSChybrid (double);
  double ShootDonnanPotential(double);
```

```
  void EvaluateDonnanPotential();
  double SelfConsistentDensities(double*);
  void EvaluateIonDensities();
  void CalculateThermodynamicPotentialsOld ();
  void CalculateThermodynamicPotentials ();
  void ExtrapolateNewCorrelationFunctions(int, int, double*);
  void RememberCorrelationFunctions(int,int);
  void RecallCorrelationFunctions(int,int);
  void RecordCorrelationFunctions ();
  void RecordThermodynamicPotentials ();
  void RecordTime(int*);
  void FreeParameters();


  public:

    RDFevaluator ();
    void SetParameters (double*,Ion*,char*);
    void ResetIonParameters(bool);
    void SetMethod (char*);
    void Run (double);
    void RunAddSaltSeries (int,double);
    void RunChangePolymer (int,int,double);
    void RunAddSaltSeriesThInt(int, int, double);
    void TestgDH();
    void TestDonnanPotential();

};

RDFevaluator::RDFevaluator () {

  R = 10.0;
  N = 100;
  Ni = 4;
  r = (double*) calloc (N,sizeof(double));
  K = (double*) calloc (N,sizeof(double));
  mu_exc = (double*) calloc (Ni,sizeof(double));
  N_ads = (double*) calloc (Ni,sizeof(double));
  omega_ads = (double*) calloc (Ni,sizeof(double));
  for (k=0;k<N;k++) r[k] = k*R/double(N-1);
  for (k=0;k<N;k++) K[k] = double(k)/R;

}
```

```
void RDFevaluator::SetParameters (double *p, Ion *iontype, char *name) {

   strcpy (projectname,name);

   N               = p[0];
   lB              = p[1];
   RD              = p[2];
   Ni              = p[3];

   //ion = (Ion*) calloc (Ni, sizeof(Ion));
   ion = new Ion[Ni];

   for (i=0;i<Ni;i++) {
      ion[i].Charge = iontype[i].Charge;
      ion[i].Diameter = iontype[i].Diameter;
      ion[i].Concentration = iontype[i].Concentration;
      strcpy(ion[i].name,iontype[i].name);
   }

   kappaD = 0.0;
   for (i=0;i<Ni;i++) { kappaD += pow(ion[i].Charge,2.0) *ion[i].Concentration; }
   kappaD = sqrt(4*pi*lB*kappaD);

   Rmin = 16.0;
   R = ion[0].Diameter + RD/kappaD;   if (R < Rmin) R = Rmin;

   d.assign(0.0,Ni,Ni);
   rho.assign(0.0,Ni,Ni);
   delta.assign(0.0,Ni,Ni);
   roots.assign(0.0,3,Ni);

   phi.assign(0.0,Ni,N);

   h.assign(0.0,Ni,Ni,N);
   c.assign(0.0,Ni,Ni,N);
   hnew.assign(0.0,Ni,Ni,N);
   cnew.assign(0.0,Ni,Ni,N);
   hFt.assign(0.0,Ni,Ni,N);
   cFt.assign(0.0,Ni,Ni,N);
   hDH.assign(0.0,Ni,Ni,N);
   hLDH.assign(0.0,Ni,Ni,N);
   hMem.assign(0.0,Ni,Ni,N);
   cMem.assign(0.0,Ni,Ni,N);
   cprev.assign(0.0,Ni,Ni,N);
```

```
  f.assign(0.0,Ni,Ni,N);
  u.assign(0.0,Ni,Ni,N);
  uFT.assign(0.0,Ni,Ni,N);

  hBigMem = (double*) calloc (Ni*Ni*N,sizeof(double));
  cBigMem = (double*) calloc (Ni*Ni*N,sizeof(double));
  guessPhiBigMem = (double*) calloc (Ni*N,sizeof(double));
  hBigMem2 = (double*) calloc (Ni*Ni*N,sizeof(double));
  cBigMem2 = (double*) calloc (Ni*Ni*N,sizeof(double));
  guessPhiBigMem2 = (double*) calloc (Ni*N,sizeof(double));

  phiZeta = (double*) calloc (Ni,sizeof(double));
  rho_res = (double*) calloc (Ni,sizeof(double));
  mu_res = (double*) calloc (Ni,sizeof(double));
  guessPhi = (double*) calloc (Ni,sizeof(double));
  guessPhiMem = (double*) calloc (Ni,sizeof(double));

  r = (double*) realloc (r,(N+1)*sizeof(double));
  K = (double*) realloc (K,(N+1)*sizeof(double));
  for (k=0;k<N;k++) r[k] = (k+1)*R/double(N+1);
  for (k=0;k<N;k++) K[k] = double(k+1)/R;

  mu_exc = (double*) realloc (mu_exc,(Ni+1)*sizeof(double));

  FirstRecording = true; FastRun = true; evaluations = 0;
  tolerance = 1e-5; //1e-5
  Nres = Ni;

  Gamma = 0; for (i=0;i<Ni;i++) if (Gamma <
ion[i].Charge*ion[i].Charge*lB/ion[i].Diameter) Gamma =
ion[i].Charge*ion[i].Charge*lB/ion[i].Diameter;
  /* One can prove that the maximal Gamma is always found between two ions of the
same type */
  alpha = 0.01; turn=1.0; turnstep = 1e-5;

  if(Gamma > 312.0){
    alpha = 0.1 / (1.0 + 0.5*Gamma);
    cout << "alpha = " << alpha << " for Gamma = " << Gamma << endl;
  }


  for (i=0;i<Ni;i++) {    delta[i][i] = 1.0;         }

}
```

```
void RDFevaluator::ResetIonParameters(bool SetLocalChargeNeutrality){

  double chargeneutral=0, Nd, totalrho;
  bool Mem;

  Nd = 20; // refines the grid on the length scale of the particle size; set 0 to turn of and
create an equidistant grid

  Mem = SetLocalChargeNeutrality;

  for (i=0;i<Ni;i++) chargeneutral += ion[i].Concentration * ion[i].Charge;
  if (chargeneutral != 0) {
    cout << "warning, the system is not charge neutral, but has an excess charge of " <<
chargeneutral << " e / nm^3" << endl;
    if (chargeneutral > 1e-6) SetLocalChargeNeutrality = true;
    if(SetLocalChargeNeutrality) {
      ion[0].Concentration -= chargeneutral/ion[0].Charge;
      cout << "Bulk concentration of ion type 1 (" << ion[0].name << ") adapted to a new
value of " << ion[0].Concentration/NA << " M" << endl;
      if(ion[0].Concentration < 0) cout << "which is negative (!). Program does not know
how to correct.";
      chargeneutral=0;
      for (i=0;i<Ni;i++) chargeneutral += ion[i].Concentration * ion[i].Charge;
      cout << "total charge is now " << chargeneutral << " e / nm^3" << endl;
    }
  }

  SetLocalChargeNeutrality = Mem;


  kappaD = 0.0;
  for (i=0;i<Ni;i++) { kappaD += pow(ion[i].Charge,2.0) *ion[i].Concentration; }
  kappaD = sqrt(4*pi*lB*kappaD);

  R = ion[0].Diameter + RD/kappaD;   if (R < Rmin) R = Rmin;

  for (k=0;k<N;k++) r[k] = (k+1)*R/double(N+1);
  for (k=0;k<N;k++) K[k] = double(k+1)/R;

  //ShiftGrid();

  Gamma = 0; for (i=0;i<Ni;i++) if (Gamma <
ion[i].Charge*ion[i].Charge*lB/ion[i].Diameter) Gamma =
```

```
ion[i].Charge*ion[i].Charge*lB/ion[i].Diameter;

   totalrho=0;
   for (i=0;i<Ni;i++) { totalrho += ion[i].Concentration; }
   if (totalrho/NA > 0.05) { alpha = 0.01; turn=0.1; FastRun = false;}
   if (totalrho/NA <= 0.05) { alpha = 0.04; turn=1.0; FastRun = true; }
   alpha = 0.01; turn=1.0; FastRun = true; //ALERT this will do a fast run, without
annealing. The memory functions may be able to replace the annealing procedure
   //alpha = 0.01; turn=1.0; FastRun = false;

}

void RDFevaluator::ShiftGrid(){

   int kcontact;

   k=0; while (r[k] < d[0][0]) k++;
   kcontact = k-1;
   if(kcontact>-1) {
      R *= (d[0][0]-1e-4)/r[kcontact];
      //R *= (d[0][0]+1e-4)/r[20];

      cout << "Grid shifted by a factor of " << d[0][0]/r[kcontact] << endl;

      cout << "before r[" << kcontact << "] = " << r[kcontact] << endl;
      for (k=0;k<N;k++) r[k] = (k+1)*R/double(N+1);
      for (k=0;k<N;k++) K[k] = double(k+1)/R;
      cout << "after r[" << kcontact << "] = " << r[kcontact] << endl;
   }


}

void RDFevaluator::SetPotential(bool HSC){

   int kd;

   for (i=0;i<Ni;i++) {
      for (j=0;j<Ni;j++) {
         d[i][j] = 0.5 * (ion[i].Diameter + ion[j].Diameter);
         rho[i][j] = sqrt(ion[i].Concentration*ion[j].Concentration);
      }
   }
```

```
  if (!HSC) {
    for (i=0;i<Ni;i++) {
      for (j=0;j<Ni;j++) {
        for (k=0;k<N;k++) {
          u[i][j][k] = 0.0;
          uFT[i][j][k] = 0.0;
        }
      }
    }
  } else {
    for (i=0;i<Ni;i++) {
      for (j=0;j<Ni;j++) {
        k=0; while (r[k] < d[i][j]) k++;
        kd=k;
        for (k=0;k<N;k++) {
          if (r[k] >= d[i][j]) u[i][j][k] = ion[i].Charge * ion[j].Charge *lB/ r[k] ;
          u[i][j][k] = ion[i].Charge * ion[j].Charge *lB/ r[k] ;
          uFT[i][j][k] = ion[i].Charge * ion[j].Charge *lB   * 4.0 / (pi*K[k]*K[k]) *
(cos(pi*K[k]*r[0]) - cos(pi*K[k]*R)) * rho[i][j];
          //uFT[i][j][k] = ion[i].Charge * ion[j].Charge *lB   * 4.0 / (pi*K[k]*K[k]) *
(cos(pi*K[k]*r[kd]) - cos(pi*K[k]*R)) * rho[i][j];
        }
      }
    }
  }




}

void RDFevaluator::SetMethod (char *appr){

  method = 6; strcpy(methodname,"MSA");
  if (strcmp(appr,"HNC")==0) { method = 1; strcpy(methodname,"HNC"); }
  if (strcmp(appr,"DHEMSA")==0) { method = 2; strcpy(methodname,"DHEMSA"); }
  if (strcmp(appr,"MSA")==0) { method = 3; strcpy(methodname,"MSA"); }
  if (strcmp(appr,"PY")==0) { method = 4; strcpy(methodname,"PY"); }

  cout << endl;
  switch (method) {

    case 1:
      cout << "Method applies the hyper-netted chain (HNC) closure" << endl;
```

```
        break;
      case 2:
        cout << "Method applies the Debye Hueckel extended mean spherical
approximation (DHEMSA)" << endl; /* JCP 2011 */
        break;
      case 3:
        cout << "Method applies the mean spherical approximation (MSA)" << endl;
        break;
      case 4:
        cout << "Method applies the Percus-Yevick closure (PY)" << endl;
        break;
      default:
        cout << "Method applies the mean spherical approximation (MSA)" << endl;

  }

}

double RDFevaluator::Qtot(double Phi, double rad, int l) {

  double Q=0;

  for (i=0;i<Ni;i++) {
    if (rad < d[l][i]) continue;
    Q -= 4*pi*lB* rad * ion[i].Concentration * ion[i].Charge * exp(-ion[i].Charge *
Phi/rad);
  }

  return Q;

}

double RDFevaluator::ShootPhiDHEMSA (int kA, double phiA, int l) {

  double Dphi,DphiG,phiG,phi0, Error, minr=1e9, large = 40.0, step;

  for (i=0;i<Ni;i++) {      if (minr > d[l][i]) minr = d[l][i];   }

  k = kA; phi[l][k] = phiA;
  Dphi = -kappaD*phiA;

  step = r[k]-r[k-1];

  while(r[k-1] > minr) {
```

```
      phi0 = phi[I][k];
      /* Euler step */
      DphiG = Dphi - Qtot(phi0,r[k],I) * step;              // phiG'(k-1) = phi'(k) - step *
Qtot[phi(k)]
      phiG = phi[I][k] - Dphi * step;                       // phiG(k-1)   = phi(k)   - step *
phi'(k)

      /* Corrector step */
      phi[I][k-1] = phi[I][k] - 0.5 * step * (Dphi + DphiG);                                  //
phi(k-1) = phi(k) - step * 0.5 * (phi'(k) + phiG'(k))
      Dphi = Dphi             - 0.5 * step * (Qtot(phi0,r[k],I) + Qtot(phiG,r[k-1],I));       //
phi'(k-1) = phi'(k) - step * 0.5 * (Qtot[phi(k)] + Qtot[phiG(k-1)])

      if (fabs(phi[I][k-1]) > large) {
         cout << "Large values of phi" << endl;
         Error = Dphi/(r[k]) - phi[I][k]/pow(r[k],2.0) + ion[I].Charge*lB/ pow(r[k],2.0);
         Error += large * (r[k] - minr) * Error/fabs(Error);
         return Error;
      }
      k--;
   }

   step = r[k] - minr;
   phi0 = phi[I][k];
   /* Euler step */
   DphiG = Dphi - Qtot(phi0,r[k],I) * step;
   phiG = phi[I][k] - Dphi * step;

   /* Corrector step */
   phiZeta[I] = phi[I][k] - 0.5 * step * (Dphi + DphiG);
   Dphi = Dphi             - 0.5 * step * (Qtot(phi0,r[k],I) + Qtot(phiG,minr,I));

   phiZeta[I] /= minr;

   Error = Dphi / minr - phiZeta[I] / minr + ion[I].Charge*lB/ (minr*minr); // Boundary
condition at contact (Gauss' law)

   return Error;

}

void RDFevaluator::EvaluatePhiDHEMSA () {
```

```
  double root[3], fr[2], minr = 1e9, maxr = 0, error=1e9;
  int kA, ni, iterations=0;


  for (ni=0;ni<Ni;ni++) {

    for (j=0;j<Ni;j++) {     if (minr > d[ni][j]) minr = d[ni][j];       if (maxr < d[ni][j]) maxr
= d[ni][j];    }

    /* Shooting from phi(kA) = phiA to phi at contact */

    k=0; while (r[k] < maxr + 4.0/kappaD and k < N-1) k++;
    kA = k;

    /* Initial guess for phiA */

    //root[0] = ion[ni].Charge * 0.0 * exp(-4.0);
    root[0] = 0.9*guessPhi[ni];
    root[1] = 0.99*guessPhi[ni];

    fr[0] = ShootPhiDHEMSA (kA, root[0], ni);
    fr[1] = ShootPhiDHEMSA (kA, root[1], ni);

    while (fabs(error) > tolerance) {

      if ( root[1] == root[0] ) {
        //cout << "ALERT, roots are the same, root[0] = root[1] = " << root[0] << endl;
        //wait_for_key();
      }

      if ( (fr[1]-fr[0]) != 0 ) {
        root[2] = root[1] - fr[1]*(root[1]-root[0]) / (fr[1]-fr[0]);
      }
      if ( (fr[1]-fr[0]) == 0 ) {
        //cout << "ALERT, the rootfinder found a slope of zero" << endl;
        root[2] = 0.9*root[1];
        //root[2] = root[1] - fr[1]*(root[1]-root[0]) / (fr[1]-fr[0]);
      }

      error = ShootPhiDHEMSA (kA, root[2], ni);
      //cout << "error of rootfinder (EvaluatePhiDHEMSA) = " << error << endl;

      root[0] = root[1]; root[1] = root[2]; fr[0] = fr[1]; fr[1] = error;
      if(fabs(error) > 1e11) { cout << "error exceeded a maximum value in
```

```
EvaluatePhiDHEMSA" << endl; }

        iterations++;
        if (iterations > 1e5 and error > tolerance) {
          cout << "Warning, EvaluatePhiDHEMSA does not converge fast. Iterations are
finished with an error larger than the tolerance" << endl;
          error = 0;
        }
      }

      guessPhi[ni] = root[2];

      /* Setting the final potential and h_DH and h_LDH*/

      for (k=1;k<kA+1;k++) {
        if (r[k] > minr ) {
          phi[ni][k] = phi[ni][k] / (r[k]);
          for (j=0;j<Ni;j++) hDH[j][ni][k] = exp(-ion[j].Charge*phi[ni][k]) - 1.0;
          for (j=0;j<Ni;j++) hLDH[j][ni][k] = -ion[ni].Charge*ion[j].Charge*lB* exp ((minr-
r[k])*kappaD) / r[k] / (1.0 + kappaD*minr);
        }
        for (j=0;j<Ni;j++) if (r[k] < d[j][ni] ) { hDH[j][ni][k] = hLDH[j][ni][k] = -1.0; }
      }
      for (k=kA+1;k<N;k++) {
        phi[ni][k] = phi[ni][kA] * exp ((r[kA]-r[k])*kappaD) *r[kA] / r[k];
        for (j=0;j<Ni;j++) hDH[j][ni][k] = exp(-ion[j].Charge*phi[ni][k]) - 1.0;
        for (j=0;j<Ni;j++) hLDH[j][ni][k] = -ion[ni].Charge*ion[j].Charge*lB* exp ((minr-
r[k])*kappaD) / r[k] / (1.0 + kappaD*minr);
      }
      error = 1e9;
    }

}


void RDFevaluator::EvaluateHNCSChybrid (const double Itermax) {

    double *fAid,*Reg, var,vol,error;
    double scaleSTf,scaleSTb;
    int iter=0;
    gsl_vector * vec1 = gsl_vector_alloc (Ni),* vec2 = gsl_vector_alloc (Ni);
    gsl_matrix * m = gsl_matrix_alloc (Ni, Ni);
    gsl_permutation * p = gsl_permutation_alloc (Ni);
    int sign[1], kmax;
```

```
  char ErrFunc[7];
  bool PlotCorrelationFunctions = false;

  Gnuplot g1, g2;
  vector<double> z,fz,fk;

  fAid = (double*) calloc (N,sizeof(double));

  fftw_plan psine;

  psine = fftw_plan_r2r_1d(N, fAid, fAid, FFTW_RODFT00, FFTW_MEASURE);

  itermax = Itermax;

  Reg = (double*) calloc (N,sizeof(double));

  vol = 2.0*R/double(N+1);

  scaleSTf = 2*R/double(N+1);          scaleSTb = 0.25/R;

  for (k=0;k<N;k++) Reg[k] = exp(-2.0*r[k]*r[k]/R/R);


  /* Begin of iteration */

  error = 1.0;
  while ((error > tolerance and iter < itermax) or (turn < 1.0)) {
    error = 0.0;


    /* Fourier transform from h_ij(r) to h_ij(k) ( Fourier transforms are multiplied by
sqrt(rho_i*rho_j) ) */


    for (i=0;i<Ni;i++) {
      for (j=0;j<Ni;j++) {
        for (k=0;k<N;k++) {
          fAid[k] = h[i][j][k] * r[k];
        }
        fftw_execute(psine);
        for (k=0;k<N;k++) fAid[k] *=   scaleSTf*rho[i][j] / K[k];
        for (k=0;k<N;k++) {
          hFt[i][j][k] = fAid[k];
        }
```

```
    }
  }


  /* Evaluation of c_ij(k) from h_ij(k) by Ornstein-Zernike */


  for (k=0;k<N;k++){
    for (i=0;i<Ni;i++) { for (j=0;j<Ni;j++) {
      gsl_matrix_set (m, i, j, delta[i][j] + hFt[i][j][k]);
    }}
    gsl_linalg_LU_decomp(m,p,sign);
    for (j=0;j<Ni;j++){
      for (i=0;i<Ni;i++) gsl_vector_set (vec2,i,delta[i][j]);
      gsl_linalg_LU_solve(m,p,vec2,vec1);
      for (i=0;i<Ni;i++) cFt[i][j][k] = gsl_vector_get (vec1,i);
    }
  }

  for (i=0;i<Ni;i++) {    for (j=0;j<Ni;j++){          for (k=0;k<N;k++){
    cFt[i][j][k] -= delta[i][j];
    cFt[i][j][k] *= -1.0;
  }}}


  /* Fourier transform from c_ij(k) to c_ij(r) */


  for (i=0;i<Ni;i++) {
    for (j=0;j<Ni;j++) {
      for (k=0;k<N;k++) {
        fAid[k] = cFt[i][j][k] * K[k];
      }
      fftw_execute(psine);
      for (k=0;k<N;k++) fAid[k] *= scaleSTb / (rho[i][j] * r[k]);
      for (k=0;k<N;k++) {
        cnew[i][j][k] = fAid[k];
      }
    }
  }


if (iter%1000 == 0 and PlotCorrelationFunctions) {
  try
```

```
      {
        g2.reset_plot();      g2.unset_grid();   g2.set_style("lines");   g2.cmd("set border
linewidth 2.0");

        for (k=0;k<N;k++) {        z.push_back(r[k]);            fz.push_back(0*c[0][1][k]);
fk.push_back(c[0][1][k]-cnew[0][1][k]);}
        g2.plot_xy(z,fz,"c_+-(r)");
        g2.plot_xy(z,fk,"cnew_+-(r)");

        z.clear();         fz.clear();         fk.clear();

      }
      catch (GnuplotException ge)
      {
        cout << ge.what() << endl;
      }
    }

     /* Correcting c_ij(r) for r > r_contact */

     for (k=0;k<N;k++) {
       for (i=0;i<Ni;i++) { for (j=0;j<Ni;j++) {

        var = alpha*cnew[i][j][k] + (1.0-alpha)*cprev[i][j][k];

        if (r[k]/d[i][j] >= 1.0) {
          switch (method) {

            case 1:
              cnew[i][j][k] =-turn*u[i][j][k]*Reg[k] + h[i][j][k] - log(h[i][j][k]+1.0);
//HNC
              break;
            case 2 :
              cnew[i][j][k] = - turn*u[i][j][k]*Reg[k] + pow(hDH[i][j][k]+1.0,turn)-1.0
- turn*log(hDH[i][j][k]+1.0); //DHEMSA (JCP 2011)
              break;
            case 3:
              cnew[i][j][k] = - turn*u[i][j][k]*Reg[k]; //MSA
              break;
            case 4:
              cnew[i][j][k] = (h[i][j][k] + 1.0) * (1.0 - exp(u[i][j][k])); //PY
              break;
            default:
              cnew[i][j][k] = - turn*u[i][j][k]*Reg[i]; //MSA
```

```
            }

          }

          /* Determining the error and establishing the new c_ij(r) */

          if (r[k]/d[i][j] >= 0.0 and error < fabs(c[i][j][k]-cnew[i][j][k])) { //WARNING:
the error is not checked for r < d
            error = fabs(c[i][j][k]-cnew[i][j][k]);
            kmax = k; strcpy(ErrFunc, "direct");
          }
          if(r[k]/d[i][j] >= 1.0) {
            c[i][j][k] = (1.0-alpha)*cprev[i][j][k] + alpha*cnew[i][j][k];
            if(FastRun) c[i][j][k] = cnew[i][j][k];
            if (iter<0) {
              //cout << "₩nTEST" << endl;
              c[i][j][k] = cnew[i][j][k];
            }
          } else {
            c[i][j][k] = var;
          }
          cprev[i][j][k] = c[i][j][k];
        }}
      }


      /* Fourier transform of c_ij(r) to c_ij(k) */


      for (i=0;i<Ni;i++) {
        for (j=0;j<Ni;j++) {
          for (k=0;k<N;k++) {
            fAid[k] = c[i][j][k] *r[k]; //fAid[k] = (c[i][j][k] + u[i][j][k]) *r[k];
          }
          fftw_execute(psine);
          for (k=0;k<N;k++) fAid[k] *= scaleSTf*rho[i][j] / K[k];
          for (k=0;k<N;k++) {
            cFt[i][j][k] = fAid[k]; //cFt[i][j][k] = fAid[k] - 0*uFT[i][j][k];
          }
        }
      }


      /* Evaluation of h_ij(k) from c_ij(k) by Ornstein-Zernike */
```

```
    for (k=0;k<N;k++){
      for (i=0;i<Ni;i++) { for (j=0;j<Ni;j++) {
        gsl_matrix_set (m, i, j, delta[i][j] - cFt[i][j][k]);
      }}
      gsl_linalg_LU_decomp(m,p,sign);
      for (j=0;j<Ni;j++){
        for (i=0;i<Ni;i++) gsl_vector_set (vec2,i,delta[i][j]);
        gsl_linalg_LU_solve(m,p,vec2,vec1);
        for (i=0;i<Ni;i++) hFt[i][j][k] = gsl_vector_get (vec1,i);
      }
    }

    for (i=0;i<Ni;i++) {   for (j=0;j<Ni;j++){          for (k=0;k<N;k++){
      hFt[i][j][k] -= delta[i][j];
    }}}


    /* Fourier transform from h_ij(k) to h_ij(r) */


    for (i=0;i<Ni;i++) {
      for (j=0;j<Ni;j++) {
        for (k=0;k<N;k++) {
          fAid[k] = hFt[i][j][k] * K[k];
        }
        fftw_execute(psine);
        for (k=0;k<N;k++) fAid[k] *= scaleSTb / (rho[i][j] * r[k]);
        for (k=0;k<N;k++) {
          hnew[i][j][k] = fAid[k];
        }
      }
    }


   if (iter%1000 == 0   and PlotCorrelationFunctions) {
     try
     {
       g1.reset_plot();      g1.unset_grid();   g1.set_style("lines");   g1.cmd("set border
linewidth 2.0");

        for (k=0;k<N;k++) {         z.push_back(r[k]);            fz.push_back(0*h[0][1][k]);
fk.push_back(h[0][1][k]-hnew[0][1][k]);}
```

```
        g1.plot_xy(z,fz,"h_+-(r)");
        g1.plot_xy(z,fk,"hnew_+-(r)");

        z.clear();        fz.clear();        fk.clear();

      }
      catch (GnuplotException ge)
      {
        cout << ge.what() << endl;
      }
    }

    /* Determining the error and establishing the new h_ij(r) */


    for (i=0;i<Ni;i++) {
      for (j=0;j<Ni;j++) {
        for (k=0;k<N;k++) {
          var = hnew[i][j][k];
          if (r[k]/d[i][j] < 1.0) {
            hnew[i][j][k] = -1.0;
          }
          if (r[k]/d[i][j] >= 0.0 and error < fabs(var-hnew[i][j][k])) { //WARNINg: the
error is not checked for r < d
            error = fabs(var-hnew[i][j][k]);
            kmax = k; strcpy(ErrFunc, "total");
          }
          h[i][j][k] = (1.0-alpha)*var + alpha*hnew[i][j][k];
          if(FastRun) h[i][j][k] = hnew[i][j][k];
          //if(k>N/2) h[i][j][k] = 0.0;
        }
      }
    }

    if (iter%1000 == 0) cout << "error = " << error << " at r_" << kmax << " = " << r[kmax]
<< " in the " << ErrFunc << " correlation function; at iteration " << iter << " and turn = " <<
turn << endl;
    if (iter%1000 == 0) {
      RecordCorrelationFunctions();
    }
    if (turn < 1.0) {turn += turnstep; if(turn > 1.0) turn = 1.0;}
    iter++;
  }
```

```
  /* end_of_iteration */

  cout << "error in EvaluateHNCSChybrid= " << error << " at iteration " << iter << endl;
  if (i == itermax) cout << "Maximum number of iterations reached." << endl;

  evaluations += iter;

  RecordCorrelationFunctions();

  free (fAid); free(Reg);

  gsl_vector_free (vec1); gsl_vector_free (vec2); gsl_matrix_free (m); gsl_permutation_free
(p);

  fftw_destroy_plan(psine);

}

double RDFevaluator::ShootDonnanPotential(double root) {

  double chargeneutral = 0;

  for(i=0;i<Nres;i++) chargeneutral += ion[i].Charge*exp(-ion[i].Charge*root + mu_res[i]
- mu_exc[i]);

  for(i=Nres;i<Ni;i++) chargeneutral += ion[i].Charge*ion[i].Concentration;

  return chargeneutral;

}

void RDFevaluator::EvaluateDonnanPotential () {

  double root[3], fr[2], error=1e9;

  /* Initial guess for phiA */

  root[0] = 1;
  root[1] = -1;

  fr[0] = ShootDonnanPotential (root[0]);
  fr[1] = ShootDonnanPotential (root[1]);

  while (fabs(error) > tolerance) {
```

```
      if ( (fr[1]-fr[0]) != 0 ) {
        root[2] = root[1] - fr[1]*(root[1]-root[0]) / (fr[1]-fr[0]);
      }
      if ( (fr[1]-fr[0]) == 0 ) {
        cout << "ALERT, the rootfinder found a slope of zero in EvaluateDonnanPotential"
<< endl;
        root[2] = 0.9*root[1];
        //root[2] = root[1] - fr[1]*(root[1]-root[0]) / (fr[1]-fr[0]);
      }

      error = ShootDonnanPotential (root[2]);
      //cout << "error of rootfinder (EvaluateDonnanPotential) = " << error << endl;

      root[0] = root[1]; root[1] = root[2]; fr[0] = fr[1]; fr[1] = error;
      if(fabs(error) > 1e11) { cout << "error exceeded a maximum value in
EvaluateDonnanPotential" << endl; }

    }

    phiD = root[2];
    cout << "₩nDonnan potential phiD = " <<   phiD << endl;

}

double RDFevaluator::SelfConsistentDensities(double *root) {

    double chargeneutral, MaxError;
    int itermax = 1e5;


    for(i=0;i<Nres;i++) {
      ion[i].Concentration = root[i];
    }
    chargeneutral=0;
    for (i=0;i<Ni;i++) chargeneutral += ion[i].Concentration * ion[i].Charge;
    root[Nres-1] = ion[Nres-1].Concentration - chargeneutral/ion[Nres-1].Charge;
    if (root[Nres-1] < 1e-4*root[0]) {
      root[Nres-1] = 1e-4*root[0];
      cout << "Alert: the concentration of the " << ion[Nres-1].name << " (type " << Nres
<< ") is adapted to avoid a low or negative value" << endl;
    }
    ion[Nres-1].Concentration = root[Nres-1];
```

```
  ResetIonParameters(!ImposeLocalChargeNeutrality);
  SetPotential(HSC);
  if(method==2)    EvaluatePhiDHEMSA();
  EvaluateHNCSChybrid(itermax);
  CalculateThermodynamicPotentials();

  EvaluateDonnanPotential();

  for(i=0;i<Nres;i++) {
    ion[i].Concentration = exp (mu_res[i] - mu_exc[i] - ion[i].Charge * phiD);
  }

  MaxError=0;
  for(i=0;i<Nres;i++) {
    if(fabs(MaxError) < fabs(pow(root[i]-ion[i].Concentration,3.0))) MaxError =
pow(root[i]-ion[i].Concentration,3.0);
    //MaxError += pow(root[i]-ion[i].Concentration, 2.0);
  }

  return MaxError;

}

void RDFevaluator::EvaluateIonDensities() {

  double fr[3], *rootf, error=1e9, chargeneutral;

  rootf = (double*) calloc (Nres, sizeof(double));

  for(i=0;i<Nres-1;i++) {
    roots[0][i] = ion[i].Concentration;
  }
  chargeneutral=0;
  for (i=0;i<Ni;i++) chargeneutral += ion[i].Concentration * ion[i].Charge;
  roots[0][Nres-1] = ion[Nres-1].Concentration - chargeneutral/ion[Nres-1].Charge;

  for(i=0;i<Nres-1;i++) {
    roots[1][i] = ion[i].Concentration* (0.999);
  }
  chargeneutral=0;
  for (i=0;i<Ni;i++) chargeneutral += ion[i].Concentration * ion[i].Charge;
  roots[1][Nres-1] = ion[Nres-1].Concentration - chargeneutral/ion[Nres-1].Charge;

  for(i=0;i<Nres;i++) rootf[i] = roots[0][i];
```

```
   fr[0] = SelfConsistentDensities(rootf);
   cout << "initial error 1 of rootfinder (EvaluateIonDensities) = " << fr[0] << endl;

   for(i=0;i<Nres;i++) rootf[i] = roots[1][i];
   fr[1] = SelfConsistentDensities(rootf);
   cout << "initial error 2 of rootfinder (EvaluateIonDensities) = " << fr[1] << endl;

   while (fabs(error) > tolerance) {

      //for (i=0;i<Nres;i++) cout << setw(19) << "rho_" << i+1;       cout << endl;
      //for (i=0;i<Nres;i++) cout << setw(20) << ion[i].Concentration/NA; cout << endl;
      //for (i=0;i<Nres;i++) cout << setw(20) << rho_res[i]/NA;       cout << "₩trho_res" <<
endl;

      for(i=0;i<Nres-1;i++){
         if ( roots[1][i] == roots[0][i] ) {
          cout << "ALERT, roots are the same, root[0][" << i << "]" << " = root[1][" << i <<
"]" << " = " << roots[0][i] << endl;
          wait_for_key();
         }
      }

      if ( (fr[1]-fr[0]) != 0 ) {
         for(i=0;i<Nres-1;i++) {
          roots[2][i] = roots[1][i] - fr[1]*(roots[1][i]-roots[0][i]) / (fr[1]-fr[0]);
          ion[i].Concentration = roots[2][i];
         }
      }
      chargeneutral=0;
      for (i=0;i<Ni;i++) chargeneutral += ion[i].Concentration * ion[i].Charge;
      roots[2][Nres-1] = ion[Nres-1].Concentration - chargeneutral/ion[Nres-1].Charge;
      if(roots[2][i]<0) { cout << "Alert: negative concentration found in
EvaluateIonDensities" << endl; wait_for_key(); }

      if ( (fr[1]-fr[0]) == 0 ) {
         cout << "ALERT, the rootfinder found a slope of zero" << endl;
         for(i=0;i<Nres-1;i++){ roots[2][i] = 0.9*roots[1][i]; }
         wait_for_key();
         //root[2] = root[1] - fr[1]*(root[1]-root[0]) / (fr[1]-fr[0]);
      }

      for(i=0;i<Nres;i++) rootf[i] = roots[2][i];
      error = SelfConsistentDensities(rootf);
      cout << "error of rootfinder (EvaluateIonDensities) = " << error << endl;
```

```
      for(j=0;j<Nres;j++) {
        cout << "₩nIterated Density of type " << j+1 << " (" << ion[j].name << "s) :₩t";
        for(i=0;i<3;i++) cout << setw(15) << roots[i][j]/NA << " M";
      } cout << endl;

      for(i=0;i<Nres;i++) {
        roots[0][i] = roots[1][i]; roots[1][i] = roots[2][i]; fr[0] = fr[1]; fr[1] = error;
      }


      if(fabs(error) > 1e11) { cout << "error exceeded a maximum value" << endl; }
      if(error != error) { wait_for_key(); }

    }


    free (rootf);

}

void RDFevaluator::CalculateThermodynamicPotentialsOld () {

    double integrand, f1,f2, det, gcontact;
    gsl_matrix * m = gsl_matrix_alloc (Ni, Ni);
    gsl_permutation * p = gsl_permutation_alloc (Ni);
    int sign[1];

    /* The calculation of the excess chemical potential */

    for (i=0;i<Ni;i++) {
      mu_exc[i] = 0;
      for (j=0;j<Ni;j++) {
        k = 1;
        f1 = 0.5 * h[i][j][k] * (h[i][j][k] - c[i][j][k]) - c[i][j][k] - u[i][j][k];
        while (r[k+1] < d[i][j]) {
          f2 = 0.5 * h[i][j][k+1] * (h[i][j][k+1] - c[i][j][k+1]) - c[i][j][k+1] -
u[i][j][k+1];
          integrand = f1*r[k]*r[k] + f2*r[k+1]*r[k+1];
          integrand *= 2.0*pi;
          mu_exc[i] += ion[j].Concentration * (integrand) * (r[k+1]-r[k]);
          f1 = f2;
          k++;
        }
```

```
        if(r[1] < d[i][j]) {
          integrand = f1*r[k]*r[k];
          integrand *= 4.0*pi;
          mu_exc[i] += ion[j].Concentration * (integrand) * (d[i][j]-r[k]);
        }
        f2 = 0.5 * h[i][j][k+1] * (h[i][j][k+1] - c[i][j][k+1]) - c[i][j][k+1] -
u[i][j][k+1];
        integrand = f2*r[k+1]*r[k+1];
        integrand *= 4.0*pi;
        mu_exc[i] += ion[j].Concentration * (integrand) * (r[k+1] - d[i][j]);
        f1 = f2;
        k++;
        while (k<N/2) {
          f2 = 0.5 * h[i][j][k+1] * (h[i][j][k+1] - c[i][j][k+1]) - c[i][j][k+1] -
u[i][j][k+1];
          integrand = f1*r[k]*r[k] + f2*r[k+1]*r[k+1];
          integrand *= 2.0*pi;
          mu_exc[i] += ion[j].Concentration * (integrand) * (r[k+1]-r[k]);
          f1 = f2;
          k++;
        }
      }
    }

    /* The calculation of the excess free energy */

    F_exc = 0;
    for (i=0;i<Ni;i++) {
      for (j=0;j<Ni;j++) {
        k = 1;
        while (r[k] <= d[i][j]) k++;
          F_exc += ion[i].Concentration *ion[j].Concentration * pi * pow(d[i][j],3.0) / 3.0;

          f1 = - 0.5 * h[i][j][k] * h[i][j][k] - h[i][j][k] +
(h[i][j][k]+1.0)*(log(h[i][j][k]+1.0)) + (h[i][j][k]+1.0)*u[i][j][k];
          if(method == 2)          {
            f1 = - 0.5 * h[i][j][k] * h[i][j][k] - h[i][j][k] +
(h[i][j][k]+1.0)*(log(hDH[i][j][k]+1.0)) + (h[i][j][k]+1.0)*u[i][j][k];
          }
          integrand = f1*r[k]*r[k];
          integrand *= 4.0*pi;

          F_exc += 0.5*ion[i].Concentration *ion[j].Concentration * (integrand) * (r[k] -
d[i][j]);
```

```
        while (k<N/2) {
          f2 = - 0.5 * h[i][j][k+1] * h[i][j][k+1] - h[i][j][k+1] +
(h[i][j][k+1]+1.0)*(log(h[i][j][k+1]+1.0)) + (h[i][j][k+1]+1.0)*u[i][j][k+1];
          if(method == 2)          {
             f2 = - 0.5 * h[i][j][k+1] * h[i][j][k+1] - h[i][j][k+1] +
(h[i][j][k+1]+1.0)*(log(hDH[i][j][k+1]+1.0)) + (h[i][j][k+1]+1.0)*u[i][j][k+1];
          }
          integrand = f1*r[k]*r[k] + f2*r[k+1]*r[k+1];
          integrand *= 2.0*pi;

          F_exc += 0.5*ion[i].Concentration *ion[j].Concentration * (integrand) * (r[k+1] -
r[k]);
          f1 = f2;
          k++;
        }
        //F_exc += 0.5*ion[i].Concentration * mu_exc[j];
      }
    }



    for (k=1;k<N;k++) {
    integrand=0;

      for (i=0;i<Ni;i++){
        for (j=0;j<Ni;j++){
          gsl_matrix_set (m, i, j, delta[i][j] + hFt[i][j][k]);
        }
        integrand += hFt[i][i][k];
      }

      gsl_linalg_LU_decomp(m,p,sign);
      det = gsl_linalg_LU_lndet (m);
      //cout << setw(20) << "ln det (I + H)" << setw(20) << det << setw(20) << "k" <<
setw(20) << k << endl;

      integrand -= det;
      //F_exc += 0.5* (integrand) * (4*pi*k*k) /(R*R); // * (k+1 - k) * pow(2.0*R,-3.0);
      F_exc += 0.5* (integrand) * (4*pi*k*k) * (k+1 - k) * pow(R,-3.0);

    }

    /* The calculation of the osmotic pressure */
```

```
  Pi_exc = 0;
  for (i=0;i<Ni;i++) {
    //Pi_exc += ion[i].Concentration;
    for (j=0;j<Ni;j++) {
      k=1;
      while (r[k] < d[i][j]) k++;
      gcontact = (h[i][j][k] + 1.0) - (r[k] - d[i][j])*(h[i][j][k+1]-h[i][j][k])/(r[k+1]-
r[k]);

      Pi_exc += ion[i].Concentration*ion[j].Concentration *2*pi*pow(d[i][j],3.0)*
gcontact /3.0;   // contact theorem (HC interaction)
      f1 = (h[i][j][k] + 1.0) * 4.0*pi*pow(r[k],3.0) * (u[i][j][k+1] - u[i][j][k-1]) /
(r[k+1] - r[k-1]);
      Pi_exc -= ion[i].Concentration * ion[j].Concentration / 6.0 * (f1) * (r[k]-d[i][j]);
// electrostatic contribution
      while (k < N/2) {
        f2 = (h[i][j][k+1] + 1.0) * 4.0*pi*pow(r[k+1],3.0) * (u[i][j][k+2] - u[i][j][k]) /
(r[k+2] - r[k]);
        integrand = 0.5*(f1 + f2);
        Pi_exc -= ion[i].Concentration * ion[j].Concentration / 6.0 * (integrand) *
(r[k+1]-r[k]); // electrostatic contribution
        f1 = f2;
        k++;
      }
    }
  }

  gsl_matrix_free (m); gsl_permutation_free (p);

}

void RDFevaluator::CalculateThermodynamicPotentials () {

  double integrand, f1,f2, gcontact;
  double gcontact2, Rcell;


  /* The calculation of the excess chemical potential */

  for (i=0;i<Ni;i++) {
    mu_exc[i] = 0;
    for (j=0;j<Ni;j++) {
      k=0; while (r[k] < d[i][j]) k++;
```

```
        gcontact = h[i][j][k] - (h[i][j][k+1]-h[i][j][k])/(r[k+1]-r[k])*(r[k]-d[i][j]);

        gcontact = ion[j].Concentration * (gcontact+1.0) * u[i][j][k];
        f2 = ion[j].Concentration * (h[i][j][k]+1.0) * u[i][j][k];
        //integrand = gcontact*d[i][j]*d[i][j] + f2*r[k]*r[k];
        integrand = f2*r[k]*r[k];
        integrand *= 4.0*pi;

        mu_exc[i] += (integrand) * (r[k]-d[i][j]);

        f1 = f2; //k++;
        while (k<N/2) {
          f2 = ion[j].Concentration * (h[i][j][k]+1.0) * u[i][j][k];
          //integrand = f1*r[k-1]*r[k-1] + f2*r[k]*r[k];
          integrand = f2*r[k]*r[k];
          integrand *= 4.0*pi;
          mu_exc[i] += (integrand) * (r[k]-r[k-1]);
          f1 = f2;
          k++;
        }
        /* TailCorrection*/
        //mu_exc[i] += ion[j].Concentration * (h[i][j][N/2]+1.0) * 4.0*pi*lB *r[N/2] /
kappaD;
        /* while (k<N) {
          f2 = ion[j].Concentration * (h[i][j][N/2]*exp(-(r[k]-r[N/2])*kappaD)+1.0) *
u[i][j][k];
          //integrand = f1*r[k-1]*r[k-1] + f2*r[k]*r[k];
          integrand = f2*r[k]*r[k];
          integrand *= 4.0*pi;
          mu_exc[i] += (integrand) * (r[k]-r[k-1]);
          f1 = f2;
          k++;
        } */
      }
    }

    for (i=0;i<Ni;i++) { N_ads[i] = 0; omega_ads[i] = 0; }
    for (i=0;i<Ni;i++) {
      for (j=0;j<Ni;j++) {
        k=0; while (r[k] < d[i][j]) k++;

        gcontact = h[i][j][k] - (h[i][j][k+1]-h[i][j][k])/(r[k+1]-r[k])*(r[k]-d[i][j]);
```

```
      if(i==0 and j==1) omega_ads[0] = -log(gcontact+1);
      if(i==0 and j==3) omega_ads[1] = -log(gcontact+1);
      if(i==1 and j==2) omega_ads[2] = -log(gcontact+1);
      if(i==2 and j==3) omega_ads[3] = -log(gcontact+1);

      gcontact = ion[j].Concentration * (gcontact + 1.0);
      f2 = ion[j].Concentration * (h[i][j][k] + 1.0);
      integrand = gcontact*d[i][j]*d[i][j] + f2*r[k]*r[k];
      integrand *= 2.0*pi;

      gcontact = (integrand) * (r[k]-d[i][j]);

      f1 = f2;
      while (r[k]<2*d[i][j]) {
        f2 = ion[j].Concentration * (h[i][j][k] + 1.0);
        integrand = f1*r[k-1]*r[k-1] + f2*r[k]*r[k];
        integrand *= 2.0*pi;
        gcontact += (integrand) * (r[k]-r[k-1]);
        f1 = f2;
        k++;
      }
      Rcell = pow(4.0*pi*ion[i].Concentration/3.0, -1.0/3.0);
      gcontact2 = gcontact;
      while (r[k]< Rcell) {
        f2 = ion[j].Concentration * (h[i][j][k] + 1.0);
        integrand = f1*r[k-1]*r[k-1] + f2*r[k]*r[k];
        integrand *= 2.0*pi;
        gcontact2 += (integrand) * (r[k]-r[k-1]);
        f1 = f2;
        k++;
      }
      if(i==0 and j==1) N_ads[0] = gcontact/gcontact2;
      if(i==0 and j==3) N_ads[1] = gcontact/gcontact2;
      if(i==1 and j==2) N_ads[2] = gcontact/gcontact2;
      if(i==2 and j==3) N_ads[3] = gcontact/gcontact2;
    }
  }
  /*
  for (i=0;i<Ni;i++) {
    N_ads[i] = 0;
    for (j=0;j<Ni;j++) {
      k=0; while (r[k] < d[i][j]) k++;

      gcontact = h[i][j][k] - (h[i][j][k+1]-h[i][j][k])/(r[k+1]-r[k])*(r[k]-d[i][j]);
```

```
        gcontact = ion[j].Concentration * (gcontact);
        f2 = ion[j].Concentration * (h[i][j][k]);
        integrand = gcontact*d[i][j]*d[i][j] + f2*r[k]*r[k];
        //integrand = f2*r[k]*r[k];
        integrand *= 2.0*pi;

        N_ads[i] += (integrand) * (r[k]-d[i][j]);

        f1 = f2; //k++;
        while (r[k]<2*d[i][j]) {
          f2 = ion[j].Concentration * (h[i][j][k]);
          integrand = f1*r[k-1]*r[k-1] + f2*r[k]*r[k];
          //integrand = f2*r[k]*r[k];
          integrand *= 2.0*pi;
          N_ads[i] += (integrand) * (r[k]-r[k-1]);
          f1 = f2;
          k++;
        }
      }
    }*/


    /* The calculation of the excess free energy */

    F_exc = 0;
    for (i=0;i<Ni;i++) {
      for (j=0;j<Ni;j++) {
        k=0; while (r[k] < d[i][j]) k++;

        f2 = 0.5 * ion[j].Concentration * ion[i].Concentration * (h[i][j][k]+1.0) *
u[i][j][k];
        integrand = f2*r[k]*r[k];
        integrand *= 4.0*pi;

        F_exc += (integrand) * (r[k]-d[i][j]);

        f1 = f2; k++;
        while (k<N/2) {
          f2 = 0.5 * ion[j].Concentration * ion[i].Concentration * (h[i][j][k]+1.0) *
u[i][j][k];
          integrand = f1*r[k-1]*r[k-1] + f2*r[k]*r[k];
          integrand *= 2.0*pi;
          F_exc += (integrand) * (r[k]-r[k-1]);
```

```
        f1 = f2;
        k++;
       }
     }
   }

   /* The calculation of the osmotic pressure */

   Pi_exc = 0;
   for (i=0;i<Ni;i++) {
     //Pi_exc += ion[i].Concentration;
     for (j=0;j<Ni;j++) {
       k=0;
       while (r[k] < d[i][j]) k++;
       gcontact = (h[i][j][k] + 1.0) - (r[k] - d[i][j])*(h[i][j][k+1]-h[i][j][k])/(r[k+1]-
r[k]);

       Pi_exc += ion[i].Concentration*ion[j].Concentration *2*pi*pow(d[i][j],3.0)*
gcontact /3.0;   // contact theorem (HC interaction)
       f1 = (h[i][j][k] + 1.0) * 4.0*pi*pow(r[k],3.0) * (u[i][j][k+1] - u[i][j][k-1]) /
(r[k+1] - r[k-1]);
       Pi_exc -= ion[i].Concentration * ion[j].Concentration / 6.0 * (f1) * (r[k]-d[i][j]);
// electrostatic contribution
       while (k < N/2) {
        f2 = (h[i][j][k+1] + 1.0) * 4.0*pi*pow(r[k+1],3.0) * (u[i][j][k+2] - u[i][j][k]) /
(r[k+2] - r[k]);
         integrand = 0.5*(f1 + f2);
         Pi_exc -= ion[i].Concentration * ion[j].Concentration / 6.0 * (integrand) *
(r[k+1]-r[k]); // electrostatic contribution
         f1 = f2;
         k++;
       }
     }
   }

}

void RDFevaluator::ExtrapolateNewCorrelationFunctions(int n, int Ns, double *IBmem){

   double frac;

   frac = (IBmem[0]-IBmem[1]) / (IBmem[1]-IBmem[2]);

   cout << "₩nRemembering, and linear extrapolation of correlation functions₩n" << endl;
```

```
   for(i=0;i<Ni;i++) {

      guessPhi[i] = guessPhiBigMem[n + i*Ns] + (guessPhiBigMem[n + i*Ns]-
guessPhiBigMem2[n + i*Ns]) * frac;

      for(j=0;j<Ni;j++) {
         for(k=0;k<N;k++) {
           h[i][j][k] = hBigMem[n + i*Ns + j*(Ni*Ns) + k*(Ni*Ni*Ns)] + (hBigMem[n + i*Ns +
j*(Ni*Ns) + k*(Ni*Ni*Ns)] - hBigMem2[n + i*Ns + j*(Ni*Ns) + k*(Ni*Ni*Ns)]) * frac;
           c[i][j][k] = cBigMem[n + i*Ns + j*(Ni*Ns) + k*(Ni*Ni*Ns)] + (cBigMem[n + i*Ns
+ j*(Ni*Ns) + k*(Ni*Ni*Ns)] - cBigMem2[n + i*Ns + j*(Ni*Ns) + k*(Ni*Ni*Ns)]) * frac;
           cprev[i][j][k] = c[i][j][k];
         }
      }

   }

}

void RDFevaluator::RememberCorrelationFunctions(int n, int Ns){

   cout << "₩nRemembering correlation functions₩n" << endl;

   for(i=0;i<Ni;i++) {

      guessPhiBigMem2[n + i*Ns] = guessPhiBigMem[n + i*Ns];
      guessPhiBigMem[n + i*Ns] = guessPhi[i];

      for(j=0;j<Ni;j++) {
         for(k=0;k<N;k++) {
           hBigMem2[n + i*Ns + j*(Ni*Ns) + k*(Ni*Ni*Ns)] = hBigMem[n + i*Ns + j*(Ni*Ns) +
k*(Ni*Ni*Ns)];
           cBigMem2[n + i*Ns + j*(Ni*Ns) + k*(Ni*Ni*Ns)] = cBigMem[n + i*Ns + j*(Ni*Ns) +
k*(Ni*Ni*Ns)];
           hBigMem[n + i*Ns + j*(Ni*Ns) + k*(Ni*Ni*Ns)] = h[i][j][k];
           cBigMem[n + i*Ns + j*(Ni*Ns) + k*(Ni*Ni*Ns)] = c[i][j][k];
         }
      }

   }

}
```

```
void RDFevaluator::RecallCorrelationFunctions(int n, int Ns){

   cout << "₩nRecalling correlation functions₩n" << endl;

   for(i=0;i<Ni;i++) {

      guessPhi[i] = guessPhiBigMem[n + i*Ns];

      for(j=0;j<Ni;j++) {
         for(k=0;k<N;k++) {
           h[i][j][k] = hBigMem[n + i*Ns + j*(Ni*Ns) + k*(Ni*Ni*Ns)];
           c[i][j][k] = cBigMem[n + i*Ns + j*(Ni*Ns) + k*(Ni*Ni*Ns)];
           cprev[i][j][k] = c[i][j][k];
         }
      }

   }


}

void RDFevaluator::RecordCorrelationFunctions () {

   int count = 3, measure, length;
   char *dataname,line[80];
   ofstream datafile;


   dataname = (char*) calloc (strlen(projectname)+14,sizeof(char));
   strcpy(dataname,projectname); strcat(dataname,"_RDF.dat");

   //cout << "₩n₩nfile is named " << dataname << endl;

   datafile.open(dataname);

   datafile << "# Radial distribution function g and direct correlation function c, obtained
by the OZ equation with the " << methodname << " closure," << endl;
   datafile << "# for a system with characteristics:₩n" << endl;

   datafile << "#₩t ₩tnumber of components, Ni = " << setw(15) << Ni << endl << endl;
   for (i=0;i<Ni;i++) {
      length = sprintf (line, "#₩t ₩ttype %d (%ss) has charge %.2f", i+1, ion[i].name,
ion[i].Charge);
      datafile << line;
```

```
      if (i==0) measure = length;
      datafile << setw(20 + measure-length) << " ";
      datafile << "₩tdiameter ";
      datafile << setw(5) << ion[i].Diameter << " nm₩tbulk density " << setw(15) <<
ion[i].Concentration/NA << " M" << endl;
   }
   datafile << "₩n#" << setw(40) << "Bjerrum length" << setw(15) << "lB" << setw(15) <<
lB << " nm" << endl;
   datafile << "#" << setw(40) << "radius of the cell" << setw(15) << "RD" << setw(15) <<
RD << " /kappaD with minimal value of R_min = " << Rmin << " nm" << endl;
   datafile << "#" << setw(40) << "Debye length" << setw(15) << "1/kappaD" << setw(15)
<< 1.0/kappaD << " nm" << endl;
   datafile << "#" << setw(40) << "maximal coupling strength" << setw(15) << "Gamma"
<< setw(15) << Gamma << " kT" << endl;
   datafile << "#" << setw(40) << "cell width" << setw(15) << "R" << setw(15) << R << "
nm" << endl;
   datafile << "#" << setw(40) << "Mixing parameter" << setw(15) << "alpha" << setw(15)
<< alpha << endl;
   datafile << "#" << setw(40) << "Number of grid points" << setw(15) << " N" << setw(15)
<< N << endl;

   datafile << "₩n₩n" << "#" << setw(11) << "1) r₩n" << "#" << setw(11) << "2) k₩n";
   for (i=0;i<Ni;i++) {      for (j=i;j<Ni;j++) {         datafile << "#" << setw(7) << count << ")
g_" << i << j << "(r)₩n"; count++;       }}
   for (i=0;i<Ni;i++) {      for (j=i;j<Ni;j++) {         datafile << "#" << setw(7) << count << ")
c_" << i << j << "(r)₩n";          count++;         }}
   for (i=0;i<Ni;i++) {      for (j=i;j<Ni;j++) {         datafile << "#" << setw(7) << count << ")
gDH_" << i << j << "(r)₩n";     count++;         }}
   for (i=0;i<Ni;i++) {      for (j=i;j<Ni;j++) {         datafile << "#" << setw(7) << count << ")
gLDH_" << i << j << "(r)₩n";    count++;         }}
   for (i=0;i<Ni;i++) {      for (j=i;j<Ni;j++) {         datafile << "#" << setw(7) << count << ")
g_" << i << j << "(k)₩n";          count++;         }}
   for (i=0;i<Ni;i++) {      for (j=i;j<Ni;j++) {         datafile << "#" << setw(7) << count << ")
c_" << i << j << "(k)₩n";          count++;         }}
   datafile << "₩n#" << setw(19) << "1)";
   for (i=1;i<count-1;i++) datafile << setw(19) << i+1 << ")";
   datafile << "₩n₩n";
   if (N>3000) {
      for (k=1;k<N;k++) if(k%int(N*1.0/3000.0)==0){
         datafile << setw(20) << r[k] << setw(20) << K[k];
         for (i=0;i<Ni;i++) {      for (j=i;j<Ni;j++) {         datafile << setw(20) << h[i][j][k] +
1.0;      }}
         for (i=0;i<Ni;i++) {      for (j=i;j<Ni;j++) {         datafile << setw(20) << c[i][j][k];
                  }}
```

```
        for (i=0;i<Ni;i++) {      for (j=i;j<Ni;j++) {        datafile << setw(20) << hDH[i][j][k]
+ 1.0;            }}
        for (i=0;i<Ni;i++) {      for (j=i;j<Ni;j++) {        datafile << setw(20) <<
hLDH[i][j][k] + 1.0;    }}
        for (i=0;i<Ni;i++) {      for (j=i;j<Ni;j++) {        datafile << setw(20) << hFt[i][j][k];
                  }}
        for (i=0;i<Ni;i++) {      for (j=i;j<Ni;j++) {        datafile << setw(20) << cFt[i][j][k];
                  }}
        datafile << endl;
      }
    } else {
      for (k=1;k<N;k++) {
        datafile << setw(20) << r[k] << setw(20) << K[k];
        for (i=0;i<Ni;i++) {      for (j=i;j<Ni;j++) {        datafile << setw(20) << h[i][j][k] +
1.0;     }}
        for (i=0;i<Ni;i++) {      for (j=i;j<Ni;j++) {        datafile << setw(20) << c[i][j][k];
                  }}
        for (i=0;i<Ni;i++) {      for (j=i;j<Ni;j++) {        datafile << setw(20) << hDH[i][j][k]
+ 1.0;            }}
        for (i=0;i<Ni;i++) {      for (j=i;j<Ni;j++) {        datafile << setw(20) <<
hLDH[i][j][k] + 1.0;    }}
        for (i=0;i<Ni;i++) {      for (j=i;j<Ni;j++) {        datafile << setw(20) << hFt[i][j][k];
                  }}
        for (i=0;i<Ni;i++) {      for (j=i;j<Ni;j++) {        datafile << setw(20) << cFt[i][j][k];
                  }}
        datafile << endl;
      }
    }

    datafile.close();

    free(dataname);

}

void RDFevaluator::RecordThermodynamicPotentials () {

    double F_id, Pi_id, F_exc_LDH, Pi_exc_LDH, *mu_exc_LDH, totalConcentration=0, pCS,
eta, rhotot;
    int measure, length;
    char *dataname,line[80];
    ofstream datafile;

    mu_exc_LDH = (double*) calloc (Ni,sizeof(double));
```

```
  //WARNING line below about kcontact can be omitted
  int kcontact;   k=0; while (r[k]<d[0][0]) k++; kcontact = k;

  eta=0;
  for (i=0;i<Ni;i++) eta += pi*pow(ion[i].Diameter,3.0)/6.0 * ion[i].Concentration;
  rhotot=0;
  for (i=0;i<Ni;i++) rhotot += ion[i].Concentration;

  pCS = rhotot * (1.0 + eta + eta*eta - eta*eta*eta) / pow(1.0-eta,3.0);
  pCS = 0.0;
  F_exc_LDH = - pow(kappaD,3.0)/(12.0*pi);
  Pi_exc_LDH = - pow(kappaD,3.0)/(24.0*pi) / (1.0 + kappaD*ion[0].Diameter) + pCS;
  Pi_exc_LDH = - kappaD * lB * ion[0].Concentration / 6.0 + pCS;
  for (i=0;i<Ni;i++) mu_exc_LDH[i] = - 0.5 * kappaD * lB * ion[i].Charge * ion[i].Charge;
  //for (i=0;i<Ni;i++) mu_exc_LDH[i] = - pow(kappaD,3.0)/(8.0*pi) /
ion[i].Concentration;

  for (i=0;i<Ni;i++) totalConcentration += ion[i].Concentration;

  dataname = (char*) calloc (strlen(projectname)+14,sizeof(char));
  strcpy(dataname,projectname); strcat(dataname,"_EOS.dat");

  //cout << "₩n₩nfile is named " << dataname << endl;

  if(FirstRecording) {
    datafile.open(dataname);

    datafile << "# Radial distribution function g and direct correlation function c, obtained
by the OZ equation with the " << methodname << " closure," << endl;
    datafile << "# for a system with characteristics:₩n" << endl;

    datafile << "#₩t ₩tnumber of components, Ni = " << setw(15) << Ni << endl << endl;
    for (i=0;i<Ni;i++) {
      length = sprintf (line, "#₩t ₩ttype %d (%ss) has charge %.2f", i+1, ion[i].name,
ion[i].Charge);
      datafile << line;
      if (i==0) measure = length;
      datafile << setw(20 + measure-length) << " ";
      datafile << "₩tdiameter ";
      datafile << setw(5) << ion[i].Diameter << " nm₩tbulk density " << setw(15) <<
ion[i].Concentration/NA << " M" << endl;
    }
```

```
    datafile << "₩n#" << setw(40) << "Bjerrum length" << setw(15) << "lB" << setw(15)
<< lB << " nm" << endl;
    datafile << "#" << setw(40) << "radius of the cell" << setw(15) << "RD" << setw(15)
<< RD << " /kappaD with minimal value of R_min = " << Rmin << " nm" << endl;
    datafile << "#" << setw(40) << "Debye length" << setw(15) << "1/kappaD" <<
setw(15) << 1.0/kappaD << " nm" << endl;
    datafile << "#" << setw(40) << "maximal coupling strength" << setw(15) << "Gamma"
<< setw(15) << Gamma << " kT" << endl;
    datafile << "#" << setw(40) << "cell width" << setw(15) << "R" << setw(15) << R << "
nm" << endl;
    datafile << "#" << setw(40) << "Mixing parameter" << setw(15) << "alpha" <<
setw(15) << alpha << endl;
    datafile << "#" << setw(40) << "Number of grid points" << setw(15) << " N" <<
setw(15) << N << endl;
    datafile << "₩n₩n";
    for (i=0;i<Ni;i++) { datafile << "#" << setw(40) << i+1 << ") density of " <<
ion[i].name <<"s        (type " << i+1 << ")     (M)" << endl;                    }
    for (i=0;i<Ni;i++) { datafile << "#" << setw(40) << i+Ni+1 <<    ") excess chemical
potential                (kT)     of " << ion[i].name <<"s        (type " << i+1 << ")" <<
endl;    }
    for (i=0;i<Ni;i++) { datafile << "#" << setw(40) << i+2*Ni+1 << ") LDH excess
chemical potential       (kT)     of " << ion[i].name <<"s        (type " << i+1 << ")" <<
endl;    }
    datafile << "#" << setw(40) << 1 + 3*Ni << ") free energy per volume (ideal)
        (kT/V)" << endl;
    datafile << "#" << setw(40) << 2 + 3*Ni << ") free energy per volume (excess)
        (kT/V)" << endl;
    datafile << "#" << setw(40) << 3 + 3*Ni << ") free energy per volume (excess LDH)
        (kT/V)" << endl;
    datafile << "#" << setw(40) << 4 + 3*Ni << ") free energy per volume (total)
        (kT/V)" << endl;
    datafile << "#" << setw(40) << 5 + 3*Ni << ") osmotic pressure (ideal)
        (kT/V)" << endl;
    datafile << "#" << setw(40) << 6 + 3*Ni << ") osmotic pressure (excess)
        (kT/V)" << endl;
    datafile << "#" << setw(40) << 7 + 3*Ni << ") osmotic pressure (excess LDH)
        (kT/V)" << endl;
    datafile << "#" << setw(40) << 8 + 3*Ni << ") osmotic pressure (total)
        (kT/V)" << endl;
    datafile << "#" << setw(40) << 9 + 3*Ni << ") Donnan Potential
        (kT/e)" << endl;
    datafile << "#" << setw(40) << 10 + 3*Ni << ") total concentration of particles    (M)"
<< endl;
    for (i=0;i<Ni;i++) { datafile << "#" << setw(40) << 10 + 3*Ni + i + 1 <<    ") Number
```

```
of adsorbed ions of " << ion[i].name <<"s      (type " << i+1 << ")" << endl;   }
      for (i=0;i<Ni;i++) { datafile << "#" << setw(40) << 10 + 4*Ni + i + 1 <<     ") Mean
energy of separation of " << ion[i].name <<"s            (type " << i+1 << ")" << endl;   }

      datafile << endl;
      datafile << "#";
      for (i=0;i<5*Ni+10;i++) { datafile << setw(14) << i+1 << ")"; }
      datafile << endl;

      datafile.close();

      FirstRecording = false;
   }

   F_id = Pi_id = 0;
   for (i=0;i<Ni;i++) Pi_id += ion[i].Concentration;
   for (i=0;i<Ni;i++) F_id += ion[i].Concentration*(log(ion[i].Concentration) - 1.0);

   datafile.open(dataname,ios::app);

   for (i=0;i<Ni;i++) {      datafile << setw(15) << ion[i].Concentration /NA;     }

   for (i=0;i<Ni;i++) {      datafile << setw(15) << mu_exc[i];                              }

   for (i=0;i<Ni;i++) {      datafile << setw(15) << mu_exc_LDH[i];                          }

   datafile << setw(15) << F_id;
   datafile << setw(15) << F_exc;
   datafile << setw(15) << F_exc_LDH;
   datafile << setw(15) << F_id + F_exc;
   datafile << setw(15) << Pi_id;
   datafile << setw(15) << Pi_exc;
   datafile << setw(15) << Pi_exc_LDH;
   datafile << setw(15) << Pi_id + Pi_exc;
   datafile << setw(15) << phiD;
   datafile << setw(15) << totalConcentration/NA;
   for (i=0;i<Ni;i++) {      datafile << setw(15) << N_ads[i];                               }
   for (i=0;i<Ni;i++) {      datafile << setw(15) << omega_ads[i];
         }
   //datafile << setw(15) << kcontact;

   datafile << endl;

   datafile.close();
```

```
   free(dataname);

/* this part is written by HKK*/

   dataname = (char*) calloc (strlen(projectname)+14,sizeof(char));
   strcpy(dataname,projectname); strcat(dataname,"_EOS_simple.dat");

   datafile.open(dataname,ios::app);
   for (i=0;i<Ni;i++) { datafile<<setw(15)<<ion[i].Concentration/NA;
   datafile<<setw(15)<<mu_exc[i];}
   datafile<<setw(15)<<F_exc;
   datafile<<endl;
   datafile.close();


/*end of HKK part*/


   free(mu_exc_LDH); free(dataname);


}

void RDFevaluator::RecordTime (int *Time) {

   ofstream datafile;
   char filename[80];


   strcpy(filename, projectname); strcat(filename,"_RDF.dat");

   datafile.open(filename,ios::app);
   datafile << "₩n₩n₩n#" << setw(10) << evaluations << setw(40) << " evaluations
before convergence.₩n" << endl;
   datafile << "₩n#₩tData in file was generated in " << setprecision (0) << Time[0] << "
hours, " << Time[1] << " minutes, and " << Time[2] << " seconds" << endl;
   datafile << "₩n#₩tProgram had been running for " << setprecision (0) << Time[3] << "
hours, " << Time[4] << " minutes, and " << Time[5] << " seconds" << endl;
   datafile.close();


   strcpy(filename, projectname); strcat(filename,"_EOS.dat");

   datafile.open(filename,ios::app);
```

```
   datafile << "₩n₩n₩n#" << setw(10) << evaluations << setw(40) << " evaluations
before convergence.₩n" << endl;
   datafile << "₩n#₩tData in file was generated in " << setprecision (0) << Time[0] << "
hours, " << Time[1] << " minutes, and " << Time[2] << " seconds" << endl;
   datafile << "₩n#₩tProgram had been running for " << setprecision (0) << Time[3] << "
hours, " << Time[4] << " minutes, and " << Time[5] << " seconds" << endl;
   datafile.close();



   cout << "₩nprogram ran for " << setprecision (0) << Time[0] << " hours, " << Time[1]
<< " minutes, and " << Time[2] << " seconds" << setprecision (10) << endl;

}

void RDFevaluator::Run (double Itermax) {

   time_t start,end;
   int Time[6],ns;
   time (&start);
   double IB0=IB,rho0,Steps=100.0,step;

   rho0=ion[1].Concentration; ion[1].Concentration = rho0/(Steps);

   itermax = Itermax;
   HSC=true;

   for(i=0;i<Ni;i++) guessPhi[i] = ion[i].Charge * 0.1 * exp(-4.0);

   if(Gamma > 10) { Steps = 1.0; cout << "Going to IB = " << IB0 << " in " << Steps << "
steps." << endl; }
   for (ns=0;ns<Steps;ns++) {
      //IB = (ns+1)*IB0/(Steps);
      ion[1].Concentration = (ns+1)*rho0/(Steps);

      SetPotential(HSC);
      ResetIonParameters(ImposeLocalChargeNeutrality);
      //cout << "IB = " << IB << ", alpha = " << alpha << " and Gamma = " << Gamma <<
endl;
      cout << "salt concentration = " << ion[0].Concentration/NA << " M" << endl;
      turn = 1.0;
      if (ns == 0) turn = 0.1;
      if(method==2)     EvaluatePhiDHEMSA();
      EvaluateHNCSChybrid(itermax);
```

```
      CalculateThermodynamicPotentials();
    }

    RecordCorrelationFunctions();
    RecordThermodynamicPotentials();


    time (&end);    Time[0] = int(difftime (end,start)/3600.0);     Time[1] = int(int(difftime
(end,start))%3600/60.0);     Time[2] = int(difftime (end,start))%60;
    RecordTime (Time);


}

void RDFevaluator::RunAddSaltSeries (int Ns, double Itermax) {

    time_t start,end,check1,check2,check3;
    int Time[6];
    time (&start);
    double factor, test;
    int n;

    itermax = Itermax;
    HSC=true;

    time (&check1); time (&check3);

    ion[0].Concentration*=1e-7;ion[1].Concentration*=1e-7;

    for(i=0;i<Ni;i++) guessPhi[i] = ion[i].Charge * 0.1 * exp(-4.0);

    factor = pow(10.0, 7.0 / (1.0*Ns-1.0));
    for (n=0;n<Ns;n++) {

      ResetIonParameters(ImposeLocalChargeNeutrality);

      cout << "changing the concentration of the " << ion[0].name << "s (type 1), and the "
<< ion[1].name << "s (type 2)" << endl;
      ion[1].Concentration += fabs(ion[0].Charge/ion[1].Charge) * (factor - 1.0)
*ion[0].Concentration;
      ion[0].Concentration += (factor - 1.0) *ion[0].Concentration;
      cout << ion[0].name << "s (type " << 1 << ") density = " << ion[0].Concentration/NA
<< " M" << endl;
      cout << ion[1].name << "s (type " << 2 << ") density = " << ion[1].Concentration/NA
```

```
<< " M" << endl;
      test=0; for (i=0;i<Ni;i++) test += ion[i].Concentration * ion[i].Charge;
      cout << "total charge density = " << test/NA << " M" << endl;

      ResetIonParameters(ImposeLocalChargeNeutrality);
        cout << "₩n₩nR = " << R << " nm₩n₩n" << endl;
      SetPotential(HSC);
      if(method==2)     EvaluatePhiDHEMSA();
      EvaluateHNCSChybrid(itermax);
      CalculateThermodynamicPotentials();
      RecordCorrelationFunctions();
      RecordThermodynamicPotentials();

      time (&check2); if (int(difftime (check2,check3)) > 5*60) {
        Time[0] = int(difftime (check2,check1)/3600.0);     Time[1] = int(int(difftime
(check2,check1))%3600/60.0);     Time[2] = int(difftime (check2,check1))%60;
        cout << "₩nprogram has been running for " << setprecision (0) << Time[0] << "
hours, " << Time[1] << " minutes, and " << Time[2] << " seconds" << setprecision (10)
<< endl;
        time (&check3);
      }
    }

   time (&end);    Time[0] = int(difftime (end,start)/3600.0);     Time[1] = int(int(difftime
(end,start))%3600/60.0);     Time[2] = int(difftime (end,start))%60;
   RecordTime (Time);


}

void RDFevaluator::RunAddSaltSeriesThInt (int Ns, int Nl, double Itermax) {

   time_t start,end, check1,check2,check3;
   int Time[6];
   int n, nl, Nmem;
   double *Fexc,*muexc, delta_lambda, IB_final, IBmem[3], RDmem;
   double factor, test, decades, totalrho;
   char rootname[60],gamma[14];

   Fexc = (double*) calloc (Ns,sizeof(double));
   muexc = (double*) calloc (Ni*Ns,sizeof(double));

   hBigMem = (double*) realloc (hBigMem,(Ni*Ni*N*Ns+1)*sizeof(double));
   cBigMem = (double*) realloc (cBigMem,(Ni*Ni*N*Ns+1)*sizeof(double));
```

```
  guessPhiBigMem = (double*) realloc (guessPhiBigMem,(Ni*Ns+1)*sizeof(double));
  hBigMem2 = (double*) realloc (hBigMem2,(Ni*Ni*N*Ns+1)*sizeof(double));
  cBigMem2 = (double*) realloc (cBigMem2,(Ni*Ni*N*Ns+1)*sizeof(double));
  guessPhiBigMem2 = (double*) realloc (guessPhiBigMem2,(Ni*Ns+1)*sizeof(double));


  delta_lambda = 1.0/double(Nl);

  Nmem = N;
  IB_final = IB;
  RDmem = RD;

  strcpy (rootname, projectname);

  time (&start);

  for (nl=0;nl<Nl;nl++){

    IB = (nl+1) * IB_final * delta_lambda;
    IBmem[2] = IBmem[1]; IBmem[1] = IBmem[0]; IBmem[0] = IB;

    itermax = Itermax;
    HSC=true;
    FirstRecording=true;

    time (&check1); time (&check3);

    decades = 7.0;
    //ion[0].Concentration*=pow(10.0,-decades);
    ion[1].Concentration*=pow(10.0,-decades);
    factor = pow(10.0, decades / double(Ns));

    for(i=0;i<Ni;i++) guessPhi[i] = ion[i].Charge * 0.1 * exp(-4.0);
    //RememberCorrelationFunctions(0,Ns);


    for (n=0;n<Ns;n++) {

      ResetIonParameters(ImposeLocalChargeNeutrality);

      cout << "changing the concentration of the " << ion[0].name << "s (type 1), and
the " << ion[1].name << "s (type 2)" << endl;
      //ion[1].Concentration += fabs(ion[0].Charge/ion[1].Charge) * (factor - 1.0)
*ion[0].Concentration;
```

```
        //ion[0].Concentration += (factor - 1.0) *ion[0].Concentration;
        //ion[0].Concentration += fabs(ion[1].Charge/ion[0].Charge) * (factor - 1.0)
*ion[1].Concentration;
        ion[1].Concentration += (factor - 1.0) *ion[1].Concentration;

        ResetIonParameters(ImposeLocalChargeNeutrality);

        cout << ion[0].name << "s (type " << 1 << ") density = " <<
ion[0].Concentration/NA << " M" << endl;
        cout << ion[1].name << "s (type " << 2 << ") density = " <<
ion[1].Concentration/NA << " M" << endl;
        test=0; for (i=0;i<Ni;i++) test += ion[i].Concentration * ion[i].Charge;
        cout << "total charge density = " << test/NA << " M" << endl;
        cout << "Gamma = " << Gamma << " kT" << endl;

        //ion[2].Concentration = 0.1 - ion[1].Concentration - ion[0].Concentration;

        //RD = RDmem; ResetIonParameters(ImposeLocalChargeNeutrality);
        //if(ion[0].Concentration < 5e-4 and Gamma > 8.0) { RD = 3.0 * kappaD; }

        totalrho=0; for (i=0;i<Ni;i++) totalrho += 0.5*ion[i].Concentration *
fabs(ion[i].Charge)/NA;

        N = Nmem;
        if(totalrho > pow(10.0,-5)) N/= 2;
        if(totalrho > pow(10.0,-3)) N/= 2;
        if(totalrho > pow(10.0,-1)) N/= 2;

        ResetIonParameters(ImposeLocalChargeNeutrality);
        sprintf(gamma,"_Gamma_%.3f",Gamma); strcpy (projectname, rootname); strcat
(projectname, gamma);


        if(nl>0) {
          RecallCorrelationFunctions(n,Ns);
          //if(nl>1) ExtrapolateNewCorrelationFunctions(n,Ns,lBmem);
          if(totalrho > 0.1 and Gamma > 0.0) {
             alpha = 0.01/sqrt(Gamma);
             turn = double(nl)/double(nl+1); turnstep = 5e-5/sqrt(Gamma);
             //turnstep = 1e-4;
             FastRun = false;
             //FastRun = true;
          }
          if(totalrho > 1.0 and Gamma > 110.0) {
```

```
            alpha = 0.01;
          }
        }
        //FastRun = false;
        //if(nl==0) { FastRun = true; turn = 1; }
        if(nl==0 and totalrho > 0.1) {
          alpha = 0.01/sqrt(Gamma);
          turn = 1.0/factor; turnstep = 1e-4;
          FastRun = false;
        }
        //if(nl<3 and ion[0].Concentration/NA > 0.05) { FastRun = false; turn = 1.0;
turnstep = 5e-4; }
        if(Gamma < 4.0 and totalrho > 10.08) { alpha = 0.05; turn = 1; turnstep = 5e-4;
FastRun = true;}
        SetPotential(HSC);

        /* Check if the free energy still has finite values */
        if(Fexc[n]!=Fexc[n]) { F_exc = 0; for(i=0;i<Ni;i++) mu_exc[i] = 0; goto Record; }

        if(method==2)     EvaluatePhiDHEMSA();

        EvaluateHNCSChybrid(itermax); cout << "alpha = " << alpha << endl;
        CalculateThermodynamicPotentials();

        F_exc = Fexc[n] + delta_lambda*F_exc *IB_final/IB; // (u(r) needs to be the
unperturbed pair potential, hence the factor IB_final/IB)
        for(i=0;i<Ni;i++) mu_exc[i] = muexc[n + Ns*i] + delta_lambda*mu_exc[i]
*IB_final/IB;

        Fexc[n] = F_exc; for(i=0;i<Ni;i++) muexc[n + Ns*i] = mu_exc[i];

        Record:
        RecordCorrelationFunctions();
        RecordThermodynamicPotentials();


        time (&check2); if (int(difftime (check2,check3)) > 5*60) {
          Time[0] = int(difftime (check2,check1)/3600.0);         Time[1] = int(int(difftime
(check2,check1))%3600/60.0);Time[2] = int(difftime (check2,check1))%60;
          Time[3] = int(difftime (check2,start )/3600.0); Time[4] = int(int(difftime
(check2,start ))%3600/60.0);    Time[5] = int(difftime (check2,start ))%60;
          cout << "₩nGamma = " << Gamma << " series has been running for " <<
setprecision (0) << Time[0] << " hours, " << Time[1] << " minutes, and " << Time[2] << "
seconds" << setprecision (10) << endl;
```

```
          cout << "₩nProgram " << projectname << " has been running for "    <<
setprecision (0) << Time[3] << " hours, " << Time[4] << " minutes, and " << Time[5] << "
seconds" << setprecision (10) << endl;
          time (&check3);
         }

        RememberCorrelationFunctions(n,Ns);
      }

      time (&end);   Time[0] = int(difftime (end,check1)/3600.0);     Time[1] =
int(int(difftime (end,check1))%3600/60.0);     Time[2] = int(difftime (end,check1))%60;
      Time[3] = int(difftime (end,start)/3600.0);     Time[4] = int(int(difftime
(end,start))%3600/60.0);     Time[5] = int(difftime (end,start))%60;
      RecordTime (Time);
   }

   free(Fexc); free(muexc);
   FreeParameters();

}

void RDFevaluator::RunChangePolymer (int Np, int Nres_requested, double Itermax) {

   time_t start,end,check1,check2,check3;
   int Time[3], n, NiMem;
   double factor;
   char Bulk[64];

   itermax = Itermax;
   HSC=true;

   time (&start);

   Nres = Nres_requested;

   for(i=0;i<Ni;i++) guessPhi[i] = ion[i].Charge * 0.1 * exp(-4.0);

   NiMem = Ni;    Ni = Nres;

   /* Calculation of the bulk concentrations and chemical potentials (hence Ni is
temporarily set to Nres) */

   rho_res = (double*) realloc (rho_res,(Nres+1)*sizeof(double));
   mu_res = (double*) realloc (mu_res,(Nres+1)*sizeof(double));
```

```
  time (&check1);   time (&check3);
  cout << "Evaluating the reservoir concentration₩n" << endl;

  ResetIonParameters(ImposeLocalChargeNeutrality);
  SetPotential(HSC);
  if(method==2)    EvaluatePhiDHEMSA();
  EvaluateHNCSChybrid(itermax);
  CalculateThermodynamicPotentials();

  strcpy (Bulk,projectname);   strcat (projectname,"_Res");

  RecordCorrelationFunctions();   RecordThermodynamicPotentials();

  strcpy (projectname,Bulk); FirstRecording = true;

  time (&check2);
  Time[0] = int(difftime (check2,check1)/3600.0);    Time[1] = int(int(difftime
(check2,check1))%3600/60.0);    Time[2] = int(difftime (check2,check1))%60;
  cout << "₩nReservoir evaluation required " << setprecision (0) << Time[0] << " hours, "
<< Time[1] << " minutes, and " << Time[2] << " seconds₩n" << setprecision (10) <<
endl;

  cout << "Reservoir concentrations :" << endl;
  for (i=0;i<Ni;i++) {
    cout << ion[i].name << "s (type " << i << ")₩t₩tdensity = " <<
ion[i].Concentration/NA << " M" << ",₩tmu_exc = " << mu_exc[i] << " kT" << endl;
  }     cout << endl;

  for(i=0;i<Nres;i++) {
    rho_res[i] = ion[i].Concentration;
    mu_res[i] = log(rho_res[i]) + mu_exc[i];
  }

  Ni = NiMem;

  /* Calculation of the concentrations and chemical potentials in the system with Nres
grand canonical, and Ni-Nres canonical species */

  //ion[Nres].Concentration *= 1e-3;
  for(i=Nres;i<Ni;i++) ion[i].Concentration *= 1e-4;

  factor = pow(10.0, 6.0 / (1.0*Np-1.0));
  for (n=0;n<Np;n++) {
```

```
      cout << "₩n** Running " << projectname << " **₩n" << endl;
      cout << "₩n₩n*** Step " << n+1 << " of " << Np << " ***" << endl;
      for(i=Nres;i<Ni;i++) cout << "changing the concentration of the " << ion[i].name <<
"s (type " << i+1 << ")" << endl;
      //ion[Nres].Concentration *= factor;
      for(i=Nres;i<Ni;i++) ion[i].Concentration *= factor;

      EvaluateDonnanPotential();
      for(i=0;i<Nres;i++) {      ion[i].Concentration = exp (mu_res[i] - mu_exc[i] -
ion[i].Charge * phiD);   }
      ResetIonParameters(!ImposeLocalChargeNeutrality);

      cout << "₩nNew concentrations :" << endl;
      for (i=0;i<Ni;i++) {
        cout << ion[i].name << "s (type " << i << ")₩t₩tdensity = " <<
ion[i].Concentration/NA << " M" << ",₩tmu_exc = " << mu_exc[i] << " kT" << endl;
      }      cout << endl;

      SetPotential(HSC);
      EvaluateIonDensities(); // rho_res and mu_res are the ion density and (total) chemical
potential of the reservoir
      if(method==2)     EvaluatePhiDHEMSA();
      EvaluateHNCSChybrid(itermax);
      CalculateThermodynamicPotentials();
      RecordCorrelationFunctions();
      RecordThermodynamicPotentials();

      time (&check2); if (int(difftime (check2,check3)) > 5*60) {
        Time[0] = int(difftime (check2,check1)/3600.0);     Time[1] = int(int(difftime
(check2,check1))%3600/60.0);     Time[2] = int(difftime (check2,check1))%60;
        cout << "₩nprogram has been running for " << setprecision (0) << Time[0] << "
hours, " << Time[1] << " minutes, and " << Time[2] << " seconds" << setprecision (10)
<< endl;
        time (&check3);
      }
    }



  time (&end);   Time[0] = int(difftime (end,start)/3600.0);     Time[1] = int(int(difftime
(end,start))%3600/60.0);     Time[2] = int(difftime (end,start))%60;
  RecordTime (Time);
```

```
   free(rho_res); free(mu_res);


}

void RDFevaluator::TestgDH() {

   HSC=true;

   SetPotential(HSC);
   EvaluatePhiDHEMSA();
   RecordCorrelationFunctions();

}

void RDFevaluator::TestDonnanPotential() {


   Nres = 3;

   rho_res = (double*) realloc (rho_res,(Nres+1)*sizeof(double));

   ion[3].Concentration *= 1e-3;
   ResetIonParameters(ImposeLocalChargeNeutrality);

   cout << "Reservoir concentrations :" << endl;
   for (i=0;i<Ni;i++) {
      cout << ion[i].name << "s (type " << i << ") density = " << ion[i].Concentration/NA
<< " M" << ", mu_exc = " << mu_exc[i] << " kT" << endl;
   }      cout << endl;

   for(i=0;i<Nres;i++) { rho_res[i] = ion[i].Concentration; }

   ion[3].Concentration *= 1e3;

   ResetIonParameters(ImposeLocalChargeNeutrality);

   cout << "₩nNew concentrations :" << endl;
   for (i=0;i<Ni;i++) {
      cout << ion[i].name << "s (type " << i << ") density = " << ion[i].Concentration/NA
<< " M" << endl;
   }      cout << endl;

   EvaluateDonnanPotential();
```

```
  free(rho_res);

}

void RDFevaluator::FreeParameters() {

  free(hBigMem);
  free(cBigMem);
  free(guessPhiBigMem);
  free(hBigMem2);
  free(cBigMem2);
  free(guessPhiBigMem2);
  free(phiZeta);
  free(rho_res);
  free(mu_res);
  free(guessPhi);
  free(guessPhiMem);
  free(r);
  free(K);

}



void CheckFile (char *datafilename, bool *present) {

  ifstream check;

  check.open (datafilename, ios::binary );
  check.seekg (0, ios::end);
  if (check.tellg()>1) {
    *present = true;
  } else {
    *present = false;
  }

}

void AskQuestion (char *datafilename, bool *proceed) {

  char answer;
```

```
   *proceed = true;
   cout << "File ₩"" << datafilename << "₩" already contains information. Overwrite (o) or
exit (e)?" << endl;
   answer = cin.get ();
   if (answer != 'o') *proceed = false;

}

void wait_for_key () {
#if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) || defined(__TOS_WIN__)
// every keypress registered, also arrow keys
      cout << endl << "Press any key to continue..." << endl;

      FlushConsoleInputBuffer(GetStdHandle(STD_INPUT_HANDLE));
      _getch();
#elif defined(unix) || defined(__unix) || defined(__unix__) || defined(__APPLE__)
      cout << endl << "Press ENTER to continue..." << endl;

      std::cin.clear();
      std::cin.ignore(std::cin.rdbuf()->in_avail());
      std::cin.get();
#endif
      return;
}


int main (){

   Ion *iontype;
   RDFevaluator HSC;
   int i, Ni, *N, IterMax, GridPoints, LambdaPartition;
   double *p;
   char projectname[70],dataname[83],methodname[7];
   bool Proceed=true,FilePresent;

   IterMax = 12e4; GridPoints = 7e1; LambdaPartition = 1e2; /*number of gamma values,
usually 1e2*/

   Ni = 2;
   N = (int*) calloc (Ni,sizeof(int));
   N[0] = 0; /* specify the ion types here, from the data bank */
   N[1] = 1;
   //N[2] = 2;
```

```
  //N[3] = 5;

  p = (double*) calloc (4, sizeof(double));
  iontype = (Ion*) calloc (Ni, sizeof(Ion));
  SetIonParameters(Ni,N,iontype);

  p[0] = pow(2.0,8.0);          /* number of grid points for the pair correlation functions
(2^10 is pretty good, and almost identical to 2^11)*/
  p[1] = 25.0*0.6;              /* Maximal Bjerrum length (Gamma parameter times ion
radius) in nm */

  p[2] = 6.0;                   /* cell width (minus diameter) in Debye lengths (6 works
well, with a minimum of 8 nm)*/
  p[3] = Ni;

  cout << "₩nThe particles are labeled as ";
  for (i=0;i<Ni-1;i++) { cout << iontype[i].name << "s (type " << i+1 << "), "; }   cout <<
"and "<< iontype[i].name << "s (type " << i+1 << ").₩n" << endl;


  strcpy(projectname,"Data/Session_01/Session_01");   strcpy(methodname,"DHEMSA");
  strcpy(dataname,projectname); strcat(dataname,"_RDF.dat");
  CheckFile(dataname,&FilePresent); if(FilePresent) AskQuestion(dataname,&Proceed);
  if(!Proceed) return 0;

  cout << "₩n** Running " << projectname << " **₩n" << endl;

  HSC.SetParameters(p,iontype,projectname);   HSC.SetMethod(methodname);

  //TestFourierTransform();
  //TestFourierTransformFFTW3();
  //HSC.RunAddSaltSeries(GridPoints,IterMax);
  //HSC.RunChangePolymer(GridPoints,Ni-1,IterMax);


  //HSC.Run(IterMax);
  HSC.RunAddSaltSeriesThInt (GridPoints, LambdaPartition, IterMax);



  //HSC.TestDonnanPotential();

  cout << "₩n₩n *** End of " << projectname << " *** ₩n₩n" << endl;
```

```
  free (p); free (iontype);
  return 0;
}
```

```
#include "system.h"

System::System(double LX, double LY, double f, double kappa, double T,
double pH, double epsilon, double pKa_1, double pKa_2, int seed) :
m_fraction(f)
{
  init_rng(seed);
  m_lattice = new Lattice(LX,LY);
  m_lb = 110.011739/(T*epsilon);  // 110.011739 comes from fixing the
units under the assumption that the lattice constant is 0.5nm
  m_n = m_n_plus = m_n_minus = 0;
  m_periodic = true;
  m_lattice->set_periodic(m_periodic);
  //m_lb = lb;


  m_min_q = new double[2];
  m_max_q = new double[2];

  m_kappa = kappa;
  m_T = T;
  m_pH = pH;
  m_epsilon = epsilon;


  m_pKa = new double[2];
  m_pKa[0] = pKa_1;
  m_pKa[1] = pKa_2;

  m_mu = new double[2];
  m_mu[0] =  log(10.0)*(pH - pKa_1); // - m_lb*m_kappa; // base
  //m_mu[1] =  -log(10.0)*(pH - pKa_2); // - m_lb*m_kappa;        //
acid
  m_mu[1] =  log(10.0)*(pH - pKa_2); // - m_lb*m_kappa;        // acid

}

System::System(double kappa, double T, double pH, double epsilon,
double pKa_1, double pKa_2, int seed) : m_fraction(0.0)
{
  init_rng(seed);
  m_lattice = new Lattice();
  m_lb = 110.011739/(T*epsilon);  // 110.011739 comes from fixing the
units under the assumption that the lattice constant is 0.5nm
  m_n = m_n_plus = m_n_minus = 0;
  m_periodic = false;
  m_lattice->set_periodic(m_periodic);
  //m_lb = lb;
```

```
  m_min_q = new double[2];
  m_max_q = new double[2];

  m_kappa = kappa;
  m_T = T;
  m_pH = pH;
  m_epsilon = epsilon;


  m_pKa = new double[2];
  m_pKa[0] = pKa_1;
  m_pKa[1] = pKa_2;

  m_mu = new double[2];
  m_mu[0] =  log(10.0)*(pH - pKa_1); // - m_lb*m_kappa; // base
  //m_mu[1] =  -log(10.0)*(pH - pKa_2); // - m_lb*m_kappa;        //
acid
  m_mu[1] =  log(10.0)*(pH - pKa_2); // - m_lb*m_kappa;        // acid
}


void System::initialize_same_charge(double q)
{
  cout << "Initializing single component system." << endl;
  if (q > 0.0)
  {
    m_min_q[0] = 0.0;  m_min_q[1] = 0.0;
    m_max_q[0] = q;    m_max_q[1] = 0.0;
  }
  else
  {
    m_min_q[0] = 0.0;  m_min_q[1] = q;
    m_max_q[0] = 0.0;  m_max_q[1] = 0.0;
  }
  for (int i = 0; i < m_lattice->get_size(); i++)
  {
    m_lattice->get_site(i)->q = q;
    if (q > 0.0)
      m_lattice->get_site(i)->type = 1;
    else
      m_lattice->get_site(i)->type = 2;
    m_n++;
  }
  if (q > 0.0)
  {
    m_n_plus = m_n;
    m_n_minus = 0.0;
  }
  else
  {
```

```
    m_n_minus = m_n;
    m_n_plus = 0.0;
  }
}

void System::initialize_opposite_charges(double q_plus, double
q_minus)
{
  cout << "Initializing a two component system." << endl;
  m_min_q[0] = 0.0;  m_min_q[1] = -fabs(q_minus);
  m_max_q[0] = fabs(q_plus);  m_max_q[1] = 0.0;

  for (int i = 0; i < m_lattice->get_size(); i++)
    if (!(m_lattice->get_has_type()))
    {
      if (drnd() < (1.0-m_fraction))
      {
        m_lattice->get_site(i)->q = 0.0;
        m_lattice->get_site(i)->type = 1;
        m_n++;
        m_n_plus++;
      }
      else
      {
        m_lattice->get_site(i)->q = 0.0;
        m_lattice->get_site(i)->type = 2;
        m_n++;
        m_n_minus++;
      }
    }
    else
    {
      m_lattice->get_site(i)->q = 0.0;
      m_n++;
      if (m_lattice->get_site(i)->type == 1)
        m_n_plus++;
      else
        m_n_minus++;
    }
}

double System::compute_energy(int i)
{
  return  compute_site_energy(i) + compute_interaction_energy(i);
}

double System::compute_energy_difference(int i, double dq)
{
  return  compute_site_energy_difference(i,dq) +
compute_interaction_energy_difference(i,dq);
```

```
}


double System::total_energy()
{
  double eng = 0.0;
  for (int i = 0; i < m_lattice->get_size(); i++)
    eng += compute_site_energy(i) + 0.5*compute_interaction_energy(i);
  return eng;
}

double System::average_charge()
{
  double charge = 0.0;
  for (int i = 0; i < m_lattice->get_size(); i++)
    charge += m_lattice->get_site(i)->q;
  return charge/m_lattice->get_size();
}

double System::average_charge(int type)
{
  double charge = 0.0;
  for (int i = 0; i < m_lattice->get_size(); i++)
    if (m_lattice->get_site(i)->type == type)
      charge += m_lattice->get_site(i)->q;
  if (type == 1)
  {
    if (m_n_plus == 0)
      return 0.0;
    else
      return charge/m_n_plus;
  }
  else
  {
    if (m_n_minus == 0)
      return 0.0;
    else
      return charge/m_n_minus;
  }
  //return charge/m_lattice->get_size();
}

void System::charge_distribution(double* cd)
{
  /*
  double dx = m_lattice->get_box()->Lx/Nx;
  double dy = m_lattice->get_box()->Ly/Ny;
  double x_min = m_lattice->get_box()->xlo;
  double y_min = m_lattice->get_box()->ylo;
  */
```

```
  for (int i = 0; i < m_lattice->get_size(); i++)
  {
    cd[i] += m_lattice->get_site(i)->q;
    /*
    double x = m_lattice->get_site(i)->x, y = m_lattice->get_site(i)-
>y;
    int idx_x = static_cast<int>((x - x_min)/dx);
    int idx_y = static_cast<int>((y - y_min)/dy);
    cd[idx_x][idx_y] += m_lattice->get_site(i)->q;
    */
  }
}


void System::charge_correlation(double* gr,  int site)
{
  int N = m_lattice->get_size();
  double Lx_half = 0.5*m_lattice->get_box()->Lx, Ly_half = m_lattice-
>get_box()->Ly;
  for (int i = 0; i < N; i++)
    gr[i] = 0.0;

  double qi = m_lattice->get_site(site)->q;
  for (int j = 0; j < N; j++)
  {
    double qj = m_lattice->get_site(j)->q;
    //double r = m_lattice->get_distance(site,j);
    //if (r <= Lx_half && r <= Ly_half)
    gr[j] = qi*qj;
  }

//  for (int i = 0; i < Nbins; i++)
//    gr[i] /= M_PI*((i+1)*(i+1) - i*i)*dr*dr;
}

void System::charge_correlation_type(double* gr, int Nbins, int
type_1, int type_2)
{
  int N = m_lattice->get_size();
  double dr = 0.5*m_lattice->get_box()->Lx/static_cast<double>(Nbins);
  double Lx_half = 0.5*m_lattice->get_box()->Lx, Ly_half = m_lattice-
>get_box()->Ly;
  int NN = 0;
  for (int i = 0; i < Nbins; i++)
    gr[i] = 0.0;
  for (int i = 0; i < N; i++)
  {
    if (m_lattice->get_site(i)->type == type_1)
    {
      double qi = m_lattice->get_site(i)->q;
```

```
      NN++;
      for (int j = 0; j < N; j++)
      {
        if (m_lattice->get_site(j)->type == type_2)
        {
          double qj = m_lattice->get_site(j)->q;
          double r = m_lattice->get_distance(i,j);
          if (r <= Lx_half && r <= Ly_half)
          {
            int bin = static_cast<int>(r/dr);
            gr[bin] += qi*qj;
          }
        }
      }
    }
  }
  for (int i = 0; i < Nbins; i++)
  {
    gr[i] /= static_cast<double>(NN);
    gr[i] /= M_PI*((i+1)*(i+1) - i*i)*dr*dr;
  }
}

void System::print_charges(string name, int Nx, int Ny)
{
  double ch[Nx][Ny];
  for (int i = 0; i < Nx; i++)
    for (int j = 0; j < Ny; j++)
      ch[i][j] = 0.0;

  double dx = m_lattice->get_box()->Lx/Nx;
  double dy = m_lattice->get_box()->Ly/Ny;
  double x_min = m_lattice->get_box()->xlo;
  double y_min = m_lattice->get_box()->ylo;

  for (int i = 0; i < m_lattice->get_size(); i++)
  {
    double x = m_lattice->get_site(i)->x, y = m_lattice->get_site(i)-
>y;
    int idx_x = static_cast<int>((x - x_min)/dx);
    int idx_y = static_cast<int>((y - y_min)/dy);
    ch[idx_x][idx_y] += m_lattice->get_site(i)->q;
  }

  ofstream out(name.c_str());

  for (int i = 0; i < Nx; i++)
  {
    for (int j = 0; j < Ny; j++)
      out << i << " " << j << " " << ch[i][j] << endl;
```

```
    out << endl;
  }

  out.close();
}

void System::print_energies(string name, int Nx, int Ny)
{
  double eng[Nx][Ny];
  int count[Nx][Ny];
  for (int i = 0; i < Nx; i++)
    for (int j = 0; j < Ny; j++)
    {
      eng[i][j] = 0.0;
      count[i][j] = 0;
    }

  double dx = m_lattice->get_box()->Lx/Nx;
  double dy = m_lattice->get_box()->Ly/Ny;
  double x_min = m_lattice->get_box()->xlo;
  double y_min = m_lattice->get_box()->ylo;

  for (int i = 0; i < m_lattice->get_size(); i++)
  {
    double x = m_lattice->get_site(i)->x, y = m_lattice->get_site(i)-
>y;
    int idx_x = static_cast<int>((x - x_min)/dx);
    int idx_y = static_cast<int>((y - y_min)/dy);
    eng[idx_x][idx_y] += compute_energy(i);
    count[idx_x][idx_y]++;
  }

  ofstream out(name.c_str());

  for (int i = 0; i < Nx; i++)
  {
    for (int j = 0; j < Ny; j++)
      if (count[i][j] != 0)
        out << i << " " << j << " " << eng[i][j]/count[i][j] << endl;
      else
        out << i << " " << j << " " << 0.0 << endl;
    out << endl;
  }

  out.close();
}

// private

double System::compute_site_energy(int i)
```

```
{
  Site* si = m_lattice->get_site(i);
  //double factor = 1.9046256137279145;  // sqrt(pi)/
sqrt(0.5*sqrt(3.0));
  //double self_energy = (1.2-m_kappa)*m_lb*(si->q)*(si->q);
  double self_energy = -m_kappa*m_lb*(si->q)*(si->q);
  //double self_energy = (factor-0.5*m_kappa)*m_lb*(si->q)*(si->q);
  //return m_mu[si->type-1]*fabs(si->q) + self_energy;
  return m_mu[si->type-1]*si->q + self_energy;
  //return m_mu[si->type-1]*si->q;
}

double System::compute_interaction_energy(int i)
{
  Site* si = m_lattice->get_site(i);
  Neighbors* neigh = m_lattice->get_neighbors(i);
  double q = si->q;
  double eng = 0.0;

  for (int j = 0; j < neigh->n_neighbors; j++)
  {
    double dist = neigh->neighbors[j].distance;
    eng += q*(neigh->neighbors[j].site->q)*(neigh-
>neighbors[j].energy_factor - m_lattice->get_eng_correction());
    //eng += m_lb*q*(neigh->neighbors[j].site->q)*exp(-m_kappa/dist)/
dist;
  }

  return eng;
}

double System::compute_site_energy_difference(int i, double dq)
{
  Site* si = m_lattice->get_site(i);
  //double factor = 1.9046256137279145;  // sqrt(pi)/
sqrt(0.5*sqrt(3.0));
  //double self_energy = (1.2-m_kappa)*m_lb*dq*(2.0*si->q+dq);
  double self_energy = -m_kappa*m_lb*dq*(2.0*si->q+dq);
  //double self_energy = (factor-0.5*m_kappa)*m_lb*(si->q)*(si->q);
  return m_mu[si->type-1]*dq + self_energy;
  //return m_mu[si->type-1]*si->q;
}

double System::compute_interaction_energy_difference(int i, double dq)
{
  Site* si = m_lattice->get_site(i);
  Neighbors* neigh = m_lattice->get_neighbors(i);
  double eng_diff = 0.0;

  for (int j = 0; j < neigh->n_neighbors; j++)
```

```
  {
    double dist = neigh->neighbors[j].distance;
    eng_diff += dq*(neigh->neighbors[j].site->q)*(neigh-
>neighbors[j].energy_factor - m_lattice->get_eng_correction());
    //eng += m_lb*q*(neigh->neighbors[j].site->q)*exp(-m_kappa/dist)/
dist;
  }

  return eng_diff;
}
```

```
#ifndef __SITE_H__
#define __SITE_H__

struct Site
{
  Site(int ID, double X, double Y, double Z, int TYPE, double Q) :
id(ID), x(X), y(Y), z(Z), type(TYPE), q(Q) { }
  int id;
  double x, y, z;
  int type;
  double q;
};

#endif
```

```
#include "rng.h"

#ifdef MKL_LIBRARY

const int  METHOD = 0;   //!< Method to be used
const int  BRNG =  VSL_BRNG_MCG31;  //!< Type defined in the MKL
documentation
const int RNDBUFLEN = 8192;  //!< Buffer length for the random number
sequence

double rndarray[RNDBUFLEN];  //!< Contains random numbers
VSLStreamStatePtr stream;    //!< Used internally by MKL
int rndcnt;                  //!< Counter


//! Initialize RNG (MKL)
//! \param seed initial seed for the random number generator
void init_rng(int seed)
{
  int errcode;
  errcode = vslNewStream( &stream, BRNG,  seed );
  rndcnt = 0;
}

//! Get a random number between 0 and 1 drawn from an uniform
distribution
//! \return random number between 0 and 1
double drnd()
{
  double a = 0.0;
  double b = 1.0;
  return (!rndcnt-- ? (vdRngUniform( METHOD, stream, RNDBUFLEN,
rndarray, a, b), rndcnt=RNDBUFLEN-1, rndarray[rndcnt]) :
rndarray[rndcnt]);
}

//! Get an integer random number between 0 and N drawn from an uniform
distribution
//! \param N upper bound for the interval
//! \return integer random number between 0 and N
int lrnd(int N)
{
  return static_cast<int>(N*drnd());
}

//! Free data structures used for the random number generator
//! In case of MKL library this is just a dummy function
//! as there is no need to free anything
void free_rng()
{
```

```
}

#else   // If MKL is not present fall back onto the GSL

const gsl_rng_type* GSL_RANDOM_TYPE;  //!< Pointer to the gsl_rng_type
structure which handles the RNG type
gsl_rng* GSL_RANDOM_GENERATOR;        //!< Pointer which holds the
actual random number generator


//! Initialize RNG (GSL)
//! \param seed initial seed for the random number generator
void init_rng(int seed)
{
  gsl_rng_env_setup();
  GSL_RANDOM_TYPE = gsl_rng_default;
  GSL_RANDOM_GENERATOR = gsl_rng_alloc(GSL_RANDOM_TYPE);
  gsl_rng_set(GSL_RANDOM_GENERATOR, static_cast<unsigned long
int>(seed));
}

//! Get a random number between 0 and 1 drawn from an uniform
distribution
//! \return random number between 0 and 1
double drnd()
{
  return gsl_rng_uniform(GSL_RANDOM_GENERATOR);
}

//! Get an integer random number between 0 and N drawn from an uniform
distribution
//! \param N upper bound for the interval
//! \return integer random number between 0 and N
int lrnd(int N)
{
  return static_cast<int>(N*drnd());
}

//! Free data structures used for the random number generator
//! In case of GLS library we need to free some memory
void free_rng()
{
  gsl_rng_free(GSL_RANDOM_GENERATOR);
}


#endif
```

```
#ifndef __NEIGHBOR_H__
#define __NEIGHBOR_H__

#include <list>

using namespace std;

struct Neighbor
{
  Neighbor(Site* s, double dist, double eng_fact) { site = s; distance
= dist; energy_factor = eng_fact; }
  Site* site;
  double distance;
  double energy_factor;
};

struct Neighbors
{
  Neighbors(int id) : idx(id) { }
  int idx;
  int n_neighbors;
  vector<Neighbor> neighbors;
};


#endif
```

```
#include "parse_input.h"

// Set of auxilliary functions that help parse lines of the input file
and
// reduce code bloating in the ParseInput::read_data() function.
// These are defined as statice.
// Reminder: Static non-memeber function are not visible outside the
// cpp file in which they are defined. No link object is created in
the .o file

//! Splits a string at one of the characters " ", "\t", or ","
//! \param str string to split
//! \return vector of split parts
static vector<string> split_line(const string& str)
{
  vector<string> split_string;
  split_regex(split_string, str, regex("\\s+|,|\\t"));
  return split_string;
}

// Here comes the implementation of class members

/*! Constructor extracts the Membrane object from the System object,
    sets the file name and opens appropriate file. At this stage no
    reading of the data is done.

    Constructor also sets the list of known keywords. Those are:
    "vertices", "edges", "triangles"

    \param sys Pointer to the System object
    \param file_name Name of the input file
    \throw runtime_error
*/
ParseInput::ParseInput(System* sys, const string& file_name) :
m_system(sys), m_file_name(file_name)
{
  m_lattice = sys->get_lattice();
  m_lattice->set_has_type(true);
  m_in.open(m_file_name.c_str());
  if (!m_in.is_open())
  {
        cerr << "Error opening input file : " << m_file_name << endl;
        throw runtime_error("Could not open input file.");
  }
  m_keywords.push_back("vertices");
  m_keywords.push_back("edges");
  m_keywords.push_back("triangles");

  m_current_keyword = "";   // No keywords are read yet.
  m_current_vertex_type = 1; // Start with the vertex type 1
```

```
  m_current_triangle_type = 1; // Start with triangle type 1
  m_current_line = 1;  // Set the line counter to the first line
}

/*! Does actual data reading and membrane building
    \throw runtime_error
*/
void ParseInput::read_data()
{
  ParseInput::LINE_TYPE line_type;
  // We assume that the system is a vesicle

  cout << "\t\tReading input file : " << m_file_name << endl;
  while ( getline(m_in, m_line) )
  {
    line_type = parse_line();
    if (line_type == ParseInput::KEYWORD)
      cout << "\t\t  Reading " << m_current_keyword << endl;
    if (line_type == ParseInput::DATA)
    {
      if (m_current_keyword == "vertices")
        this->parse_vertex();
      else if (m_current_keyword == "edges")
        this->parse_edge();
      else
        this->parse_triangle();
    }
    if (line_type == ParseInput::ERROR)
    {
      cerr << "Error at line " << m_current_line << " while parsing
input file " << m_file_name << "." << endl;
      throw runtime_error("Error parsing input file.");
    }
    m_current_line++;
  }
  cout << "\t\t\tRead " << m_current_line << " lines." << endl;

}


// Private method are implemented in here
/*! A line is passed as a string. It is processed and the type of the
line is returned
  \return one of the four possible line types
*/
ParseInput::LINE_TYPE ParseInput::parse_line()
{
  vector<string> s_line;
  vector<string>::iterator it_s;
  // first remove all leading and trailing spaces
```

```
  trim(m_line);
  if (m_line.size() == 0)
    return ParseInput::EMPTY;
  // transform it to a lower case
  to_lower(m_line);

  if (m_line[0] == '#')             // Check if comment
    return ParseInput::COMMENT;
  else if (isalpha(m_line[0]))      // this might be a keyword
  {
      s_line = split_line(m_line);
      if (s_line.size() != 1 && s_line[1][0] != '#')    // more than
one word in the line and the second word in not begining of a comment
--> error
        return ParseInput::ERROR;
      it_s = std::find(m_keywords.begin(), m_keywords.end(),
s_line[0]);  // check if the word is in the list of keywords
      if (it_s == m_keywords.end())   // if not --> error
        return ParseInput::ERROR;
      else
      {
        m_current_keyword = s_line[0];
        return ParseInput::KEYWORD;
      }
  }
  else if (isdigit(m_line[0]) || m_line[0] == '+' || m_line[0] == '-'
|| m_line[0] == '.')  // check if it is a number
    return ParseInput::DATA;
  else
    return ParseInput::ERROR;
}

/*! Processes line that contains vertex data
    vertex data line can have either four or five entries, id, x, y,
z, and optional type.
    If type is missing, m_current_vertex_type is used.
    \throw runtime_error
*/
void ParseInput::parse_vertex()
{
  Box* box = m_system->get_box();
  int v_type = m_current_vertex_type;
  int id;
  double x, y, z;
  vector<string> s_line = split_line(m_line);

  if (s_line.size() >= 4)  // There are four or more entries in the
line
  {
    try
```

```
    {
      id = lexical_cast<int>(s_line[0]);   // first entry in the id
    }
    catch (bad_lexical_cast &)
    {
      cerr << "Bad vertex id at line " << m_current_line << " of input
file " << m_file_name << endl;
      cerr << "  Input line : " << m_line << endl;
      throw runtime_error("Bad vertex id.");
    }
    try
    {
      x = lexical_cast<double>(s_line[1]);  // x coordinate
      y = lexical_cast<double>(s_line[2]);  // y coordinate
      z = lexical_cast<double>(s_line[3]);  // z coordinate
    }
    catch (bad_lexical_cast &)
    {
      cerr << "Bad vertex coordinate at line " << m_current_line << "
of input file " << m_file_name << endl;
      cerr << "  Input line : " << m_line << endl;
      throw runtime_error("Bad vertex coordinate.");
    }
    if (s_line.size() > 4 && s_line[4][0] != '#')  // seems that we
have type after all
    {
      try
      {
        v_type = lexical_cast<int>(s_line[4]);  // read the vertex
type
        if (find(m_v_types.begin(), m_v_types.end(), v_type) ==
m_v_types.end())
          m_v_types.push_back(v_type);
      }
      catch (bad_lexical_cast &)
      {
        cerr << "Bad vertex type at line " << m_current_line << " of
input file " << m_file_name << endl;
        cerr << "  Input line : " << m_line << endl;
        throw runtime_error("Bad vertex type.");
      }
    }
  }
  else  // line is too short
  {
    cerr << "Parse error in " << m_file_name << " at line : " <<
m_current_line << endl;
    cerr << "     Input line : " << m_line << endl;
    cerr << "     Vertex data line too short!" << endl;
    throw runtime_error("Input file parse error.");
```

```
  }

  m_current_vertex_type = v_type;

  // If we have reached up to here we heve sucessfully parsed the
vertex line
  // We can now generate a vertex
  if (m_system->get_periodic())
    if (x < box->xlo || x > box->xhi || y < box->ylo || y > box->yhi
|| z < box->zlo || z > box->zhi)
    {
      cerr << "Error reading vertex coordinates. " << endl;
      cerr << "    Simulation box too small. " << endl;
      throw runtime_error("Simulation box too small.");
    }
  m_lattice->add_site(new Site(id, x, y, z, v_type, 0.0));
}

/*! Parsing line that contains edge data. Edge data line always has
four entries:
    edge_id id_of_vertex_1 id_of_vertex_2 edge_type
*/
void ParseInput::parse_edge()
{

}

/*! Parse line that contains the triangle data. Each triangle line has
to have five or six
   entries followed by an optional commnet. Entries are:
   triangle_id id_vertex_1 id_vertex_2 id_vertex_3 orientation [type]
*/
void ParseInput::parse_triangle()
{

}
```

```
#include "parse_input.h"

// Set of auxilliary functions that help parse lines of the input file
and
// reduce code bloating in the ParseInput::read_data() function.
// These are defined as statice.
// Reminder: Static non-memeber function are not visible outside the
// cpp file in which they are defined. No link object is created in
the .o file

//! Splits a string at one of the characters " ", "\t", or ","
//! \param str string to split
//! \return vector of split parts
static vector<string> split_line(const string& str)
{
  vector<string> split_string;
  split_regex(split_string, str, regex("\\s+|,|\\t"));
  return split_string;
}

// Here comes the implementation of class members

/*! Constructor extracts the Membrane object from the System object,
    sets the file name and opens appropriate file. At this stage no
    reading of the data is done.

    Constructor also sets the list of known keywords. Those are:
    "vertices", "edges", "triangles"

    \param sys Pointer to the System object
    \param file_name Name of the input file
    \throw runtime_error
*/
ParseInput::ParseInput(System* sys, const string& file_name) :
m_system(sys), m_file_name(file_name)
{
  m_lattice = sys->get_lattice();
  m_lattice->set_has_type(true);
  m_in.open(m_file_name.c_str());
  if (!m_in.is_open())
  {
        cerr << "Error opening input file : " << m_file_name << endl;
        throw runtime_error("Could not open input file.");
  }
  m_keywords.push_back("vertices");
  m_keywords.push_back("edges");
  m_keywords.push_back("triangles");

  m_current_keyword = "";   // No keywords are read yet.
  m_current_vertex_type = 1; // Start with the vertex type 1
```

```
  m_current_triangle_type = 1; // Start with triangle type 1
  m_current_line = 1;  // Set the line counter to the first line
}

/*! Does actual data reading and membrane building
    \throw runtime_error
*/
void ParseInput::read_data()
{
  ParseInput::LINE_TYPE line_type;
  // We assume that the system is a vesicle

  cout << "\t\tReading input file : " << m_file_name << endl;
  while ( getline(m_in, m_line) )
  {
    line_type = parse_line();
    if (line_type == ParseInput::KEYWORD)
      cout << "\t\t  Reading " << m_current_keyword << endl;
    if (line_type == ParseInput::DATA)
    {
      if (m_current_keyword == "vertices")
        this->parse_vertex();
      else if (m_current_keyword == "edges")
        this->parse_edge();
      else
        this->parse_triangle();
    }
    if (line_type == ParseInput::ERROR)
    {
      cerr << "Error at line " << m_current_line << " while parsing
input file " << m_file_name << "." << endl;
      throw runtime_error("Error parsing input file.");
    }
    m_current_line++;
  }
  cout << "\t\t\tRead " << m_current_line << " lines." << endl;

}


// Private method are implemented in here
/*! A line is passed as a string. It is processed and the type of the
line is returned
  \return one of the four possible line types
*/
ParseInput::LINE_TYPE ParseInput::parse_line()
{
  vector<string> s_line;
  vector<string>::iterator it_s;
  // first remove all leading and trailing spaces
```

```
  trim(m_line);
  if (m_line.size() == 0)
    return ParseInput::EMPTY;
  // transform it to a lower case
  to_lower(m_line);

  if (m_line[0] == '#')             // Check if comment
    return ParseInput::COMMENT;
  else if (isalpha(m_line[0]))      // this might be a keyword
  {
      s_line = split_line(m_line);
      if (s_line.size() != 1 && s_line[1][0] != '#')    // more than
one word in the line and the second word in not begining of a comment
--> error
        return ParseInput::ERROR;
      it_s = std::find(m_keywords.begin(), m_keywords.end(),
s_line[0]);  // check if the word is in the list of keywords
      if (it_s == m_keywords.end())   // if not --> error
        return ParseInput::ERROR;
      else
      {
        m_current_keyword = s_line[0];
        return ParseInput::KEYWORD;
      }
  }
  else if (isdigit(m_line[0]) || m_line[0] == '+' || m_line[0] == '-'
|| m_line[0] == '.')  // check if it is a number
    return ParseInput::DATA;
  else
    return ParseInput::ERROR;
}

/*! Processes line that contains vertex data
    vertex data line can have either four or five entries, id, x, y,
z, and optional type.
    If type is missing, m_current_vertex_type is used.
    \throw runtime_error
*/
void ParseInput::parse_vertex()
{
  Box* box = m_system->get_box();
  int v_type = m_current_vertex_type;
  int id;
  double x, y, z;
  vector<string> s_line = split_line(m_line);

  if (s_line.size() >= 4)  // There are four or more entries in the
line
  {
    try
```

```
    {
      id = lexical_cast<int>(s_line[0]);   // first entry in the id
    }
    catch (bad_lexical_cast &)
    {
      cerr << "Bad vertex id at line " << m_current_line << " of input
file " << m_file_name << endl;
      cerr << "  Input line : " << m_line << endl;
      throw runtime_error("Bad vertex id.");
    }
    try
    {
      x = lexical_cast<double>(s_line[1]);  // x coordinate
      y = lexical_cast<double>(s_line[2]);  // y coordinate
      z = lexical_cast<double>(s_line[3]);  // z coordinate
    }
    catch (bad_lexical_cast &)
    {
      cerr << "Bad vertex coordinate at line " << m_current_line << "
of input file " << m_file_name << endl;
      cerr << "  Input line : " << m_line << endl;
      throw runtime_error("Bad vertex coordinate.");
    }
    if (s_line.size() > 4 && s_line[4][0] != '#')  // seems that we
have type after all
    {
      try
      {
        v_type = lexical_cast<int>(s_line[4]);  // read the vertex
type
        if (find(m_v_types.begin(), m_v_types.end(), v_type) ==
m_v_types.end())
          m_v_types.push_back(v_type);
      }
      catch (bad_lexical_cast &)
      {
        cerr << "Bad vertex type at line " << m_current_line << " of
input file " << m_file_name << endl;
        cerr << "  Input line : " << m_line << endl;
        throw runtime_error("Bad vertex type.");
      }
    }
  }
  else  // line is too short
  {
    cerr << "Parse error in " << m_file_name << " at line : " <<
m_current_line << endl;
    cerr << "     Input line : " << m_line << endl;
    cerr << "     Vertex data line too short!" << endl;
    throw runtime_error("Input file parse error.");
```

```
  }

  m_current_vertex_type = v_type;

  // If we have reached up to here we heve sucessfully parsed the
vertex line
  // We can now generate a vertex
  if (m_system->get_periodic())
    if (x < box->xlo || x > box->xhi || y < box->ylo || y > box->yhi
|| z < box->zlo || z > box->zhi)
    {
      cerr << "Error reading vertex coordinates. " << endl;
      cerr << "    Simulation box too small. " << endl;
      throw runtime_error("Simulation box too small.");
    }
  m_lattice->add_site(new Site(id, x, y, z, v_type, 0.0));
}

/*! Parsing line that contains edge data. Edge data line always has
four entries:
    edge_id id_of_vertex_1 id_of_vertex_2 edge_type
*/
void ParseInput::parse_edge()
{

}

/*! Parse line that contains the triangle data. Each triangle line has
to have five or six
   entries followed by an optional commnet. Entries are:
   triangle_id id_vertex_1 id_vertex_2 id_vertex_3 orientation [type]
*/
void ParseInput::parse_triangle()
{

}
```

```
#include "mc_step.h"


void MCStep::sweep(int swap_freq)
{
  int charge_accept = 0;
  int swap_accept = 0;
  int N = m_system->get_lattice()->get_size();

  for (int step = 0; step < N; step++)
  {
    int idx = lrnd(N);
    if (this->charge_step(idx))
      charge_accept++;
  }

  if (swap_freq > 0)
  {
    if (m_current_sweep % swap_freq == 0)
    {
      for (int step = 0; step < N; step++)
      {
        int i = lrnd(N), j = lrnd(N);
        if (this->swap_step(i,j))
          swap_accept++;
      }
    }
  }

  m_charge_accept = static_cast<double>(charge_accept)/N;
  m_swap_accept = static_cast<double>(swap_accept)/N;

  m_current_sweep++;

}

// Private methods

bool MCStep::charge_step(int i)
{

  Site* s = m_system->get_site(i);
  double q_old = s->q;

  double del_q = static_cast<double>(2*lrnd(2)-1);

  double q_new = q_old + del_q;

  if (q_new > m_system->get_max_q(s->type) || q_new <  m_system-
>get_min_q(s->type))
```

```
    return false;

  double Eold = m_system->compute_energy(i);

  s->q = q_new;

  double Enew = m_system->compute_energy(i);

  double delE = Enew - Eold;

  if (delE < 0.0)
    return true;
  else if (drnd() < exp(-delE))
    return true;

  s->q = q_old;

  return false;

/*

  Site* s = m_system->get_site(i);

  double del_q = static_cast<double>(2*lrnd(2)-1);

  double q_new = s->q + del_q;

  if (q_new > m_system->get_max_q(s->type) || q_new <  m_system-
>get_min_q(s->type))
    return false;


  double delE = m_system->compute_energy_difference(i,del_q);

  if (delE < 0.0)
  {
    s->q = q_new;
    return true;
  }
  else if (drnd() < exp(-delE))
  {
    s->q = q_new;
    return true;
  }

  return false;
*/
}

bool MCStep::swap_step(int i, int j)
```

```
{
  if (i == j) return false;
  Site* si = m_system->get_site(i);
  Site* sj = m_system->get_site(j);

  if (si->type == sj->type) return false;

  double Eold = m_system->compute_energy(i) + m_system-
>compute_energy(j);

  m_system->get_lattice()->swap_sites(i,j);

  double Enew = m_system->compute_energy(i) + m_system-
>compute_energy(j);

  double delE = Enew - Eold;

  /* Metropolis
  if (delE < 0.0)
    return true;
  else if (drnd() < exp(-delE/m_T))
    return true;
  */

  /* Kawasaki */
  if (drnd() < 1.0/(1.0+exp(delE)))
    return true;

  m_system->get_lattice()->swap_sites(i,j);

  return false;

}
```

```
#include "parse_input.h"
#include "mc_step.h"

#include <boost/format.hpp>
#include <boost/program_options.hpp>
#include <boost/lexical_cast.hpp>

#include <ctime>
#include <fstream>

using boost::format;
using namespace boost::program_options;
using boost::lexical_cast;

using namespace std;


int main(int argc, char* argv[])
{
  double L;
  int Nsteps;
  double frac;
  double T;
  double pH = 4.0;
  double epsilon;
  double lb;
  double pKa_1;
  double pKa_2;
  double inv_kappa;
  int seed;
  int swap_freq;
  int print_freq;
  string out_filename;
  string in_filename = "";
  int Neqil;
  vector<double> q;
  int measure_every;
  int Nbin;
  bool is_acid = false;
  double* gr;
  double* gr_pp;
  double* gr_mm;
  double* gr_pm;
  double* avr_gr;
  double* avr_gr_pp;
  double* avr_gr_mm;
  double* avr_gr_pm;

  char time_str[81];
  time_t start_time = time(NULL);
```

```
  tm *tmbuf=localtime(&start_time);

  strftime(time_str, 80, "# Created by pHMC code on %d %B, %Y at %H:
%M", tmbuf);

  options_description desc("Usage:\nrandom_mesh <options>");
  desc.add_options()
      ("help,h", "print usage message")
      ("box-length,L", value<double>(&L)->default_value(20.0), "box
length")
      ("in-file,i", value<string>(&in_filename), "input file name
(Membrane code input file format)")
      ("acid-frac,f", value<double>(&frac)->default_value(0.25),
"fraction of acid molecules")
      ("charges,q", value< vector<double> >(&q)->multitoken(), "space
separated charges of different components (NOTE: ALL CHARGES MUST BE
GIVEN AS POSITIVE NUMBERS, second charge will be treated as negative -
limitation of the options parsing library)")
      ("temperature,T", value<double>(&T)->default_value(1.0),
"temperature")
      ("pH,p", value<double>(&pH)->default_value(7.0), "pH")
      ("epsilon,e", value<double>(&epsilon)->default_value(78.54),
"dielectric constant of solvent")
      ("pKa_1,1", value<double>(&pKa_1)->default_value(10.5), "pKa of
base")
      ("pKa_2,2", value<double>(&pKa_2)->default_value(7.4), "pKa of
acid")
      ("screening-length,l", value<double>(&inv_kappa)-
>default_value(2.0), "screening length (in units of lattice spacing)")
      ("out-file,o", value<std::string>(&out_filename)-
>default_value("out"), "output file (extension 'out' will be appended
automatically)")
      ("mc-steps,N", value<int>(&Nsteps)->default_value(1000), "number
of Monte Carlo steps")
      ("mc-equil,E", value<int>(&Neqil)->default_value(100), "number
of equilibration Monte Carlo steps")
      ("seed,s", value<int>(&seed)->default_value(123), "random number
generator seed")
      ("print-freq,P", value<int>(&print_freq)->default_value(10),
"how often to log results")
      ("swap-freq,S", value<int>(&swap_freq)->default_value(1), "how
often to perform swap move")
      ("measure-every,m", value<int>(&measure_every)-
>default_value(10), "how often to measure quantities")
      ("bins,b", value<int>(&Nbin)->default_value(10), "number of
histogram bins for pair correlation functions")
      ("acid,A","simulate acid");

  variables_map vm;
  store(parse_command_line(argc, argv, desc), vm);
```

```
  notify(vm);

  if (vm.count("help"))
  {
    std::cout << desc << "\n";
    return 0;
  }

  if (vm.count("acid"))
    is_acid = true;

  lb = 110.011739/(T*epsilon);
  System* sys;
  if (in_filename == "")
  {
    sys = new System(L, L, frac, 1.0/inv_kappa, T, pH, epsilon, pKa_1,
pKa_2, seed);
    sys->build_neighbor_list(1.0/inv_kappa, lb);
  }
  else
  {
    sys = new System(1.0/inv_kappa, T, pH, epsilon, pKa_1, pKa_2,
seed);
    ParseInput* parse_input = new ParseInput(sys, in_filename);
    parse_input->read_data();
    sys->build_neighbor_list(1.0/inv_kappa, lb);
  }

  MCStep* mc = new MCStep(sys,seed);

  bool single_component = false;
  if (vm.count("charges"))
  {
    size_t s = vm["charges"].as< vector<double> >().size();
    if (s == 0)
    {
      cerr << "At least one charge has to be given." << endl;
      return -1;
    }
    else if (s == 1)
    {
      if (is_acid)
        sys->initialize_same_charge(-q[0]);
      else
        sys->initialize_same_charge(q[0]);
      single_component = true;
    }
    else
      sys->initialize_opposite_charges(q[0],q[1]); // first argument
is BASE, second argument is ACID
```

```
}


int N = static_cast<int>(L);

double* ch_dist;
double* avr_site_q;
ch_dist = new double[sys->get_size()];
avr_site_q = new double[sys->get_size()];
for (int i = 0; i < sys->get_size(); i++)
   ch_dist[i] = avr_site_q[i] = 0.0;



/*
for (int i = 0; i < N; i++)
{
  ch_dist[i] = new double[N];
  for (int j = 0; j < N; j++)
    ch_dist[i][j] = 0.0;
}
*/

gr        = new double[sys->get_size()];
gr_pp     = new double[Nbin];
gr_mm     = new double[Nbin];
gr_pm     = new double[Nbin];
avr_gr    = new double[Nbin];
avr_gr_pp = new double[Nbin];
avr_gr_mm = new double[Nbin];
avr_gr_pm = new double[Nbin];




int steps_to_average = 0;


cout << "# Box Length : " << L << endl;
cout << "# Number of sites : " << sys->get_size() << endl;
if (in_filename != "")
  cout << "# Reading input from file : " << in_filename << endl;
cout << "# Fraction of acid sites : " << frac << endl;
cout << "# Temperature : " << T << endl;
cout << "# pH : " << pH << endl;
cout << "# Screening length : " << inv_kappa << endl;
cout << "# Dielectric constant : " << epsilon << endl;
cout << "# pKa_base : " << pKa_1 << endl;
cout << "# pKa_acid : " << pKa_2 << endl;
cout << "# output file name : " << out_filename << endl;
```

```
  cout << "# Number of Monte Carlo steps : " << Nsteps << endl;
  cout << "# Random number generator seed : " << seed << endl;
  cout << "# Site swap frequency : " << swap_freq << endl;
  cout << "# Bjerrum length : " << lb << "  (" << 0.5*lb << "nm)" <<
endl;
  cout << "# Charges : " << endl << "\t";
  for (int qlen = 0; qlen < q.size(); qlen++)
    cout << q[qlen] << " ";
  cout << endl;
  cout << "# Number of histogram bins (pair correlation fucntion) : "
<< Nbin << endl;
  cout << "# Self energy : " << lb*(-1.0/inv_kappa) << endl;
  if (single_component)
    if (is_acid)
      cout << "# Simulating acid system. Using pKa_acid parameter. "
<< endl;
    else
      cout << "# Simulating base system. Using pKa_base parameter. "
<< endl;
  cout << "#" << endl;
  cout << time_str << endl;
  cout << "# " << endl;
  cout << "# step  total_energy  average_charge  average_charge_base
average_charge_acid  charge_change_acceptance_rate
site_swap_acceptance_rate" << endl << endl;

  double avr_charge = 0.0;
  double avr_charge_sq = 0.0;
  double avr_q_plus = 0.0, avr_q_plus_sq = 0.0, avr_q_minus = 0.0,
avr_q_minus_sq = 0.0;

  for (int i = 0; i < Nbin; i++)
  {
    avr_gr[i] = 0.0;
    avr_gr_pp[i] = 0.0;
    avr_gr_mm[i] = 0.0;
    avr_gr_pm[i] = 0.0;
  }

  double* avr_correl = new double[sys->get_size()];
  for (int site = 0; site < sys->get_size(); site++)
    avr_correl[site] = 0.0;

  for (int i = 0; i <= Nsteps; i++)
  {
    mc->sweep(swap_freq);
    if (i % print_freq == 0)
    {
      //cout <<  i << " " << sys->average_charge(2)  << endl;
      cout << (format("%6d ") % i) << " " << (format("%10.5e ") % sys-
```

```
>total_energy()) << " " << (format("%10.5e ") % sys->average_charge())
<< " " << (format("%10.5e ") % sys->average_charge(1)) << " "
           << (format("%10.5e ") % sys->average_charge(2)) << " " <<
(format("%10.5e ") % mc->get_charge_accept()) << " " <<
(format("%10.5e ") % mc->get_swap_accept()) << endl;
    }
    if (i >= Neqil && i % measure_every == 0)
      {
        //sys->charge_distribution(ch_dist, N, N);
        sys->charge_distribution(ch_dist);
        double avr_q = sys->average_charge();
        avr_charge += avr_q;
        avr_charge_sq  += avr_q*avr_q;
        for (int bin = 0; bin < Nbin; bin++)
          avr_gr[bin] += gr[bin];

        if (!single_component)
        {
          double q_p = sys->average_charge(1);
          double q_m = sys->average_charge(2);
          sys->charge_correlation_type(gr_pp, Nbin,1,1);
          sys->charge_correlation_type(gr_mm, Nbin,2,2);
          sys->charge_correlation_type(gr_pm, Nbin,1,2);
          avr_q_plus += q_p;
          avr_q_plus_sq += q_p*q_p;
          avr_q_minus += q_m;
          avr_q_minus_sq += q_m*q_m;
          for (int bin = 0; bin < Nbin; bin++)
          {
            avr_gr_pp[bin] += gr_pp[bin];
            avr_gr_mm[bin] += gr_mm[bin];
            avr_gr_pm[bin] += gr_pm[bin];
          }
        }
        sys->charge_correlation(gr, 0);
        ofstream out((out_filename+"_charges_"+lexical_cast<string>(i)
+".dat").c_str());
        for (int site = 0; site < sys->get_size(); site++)
        {
          avr_site_q[site] += sys->get_charge(site);
          avr_correl[site] += gr[site];
          out << sys->get_site(site)->x << " " << sys->get_site(site)-
>y << " " << sys->get_site(site)->type << " " << sys->get_charge(site)
<< endl;
        }
        out.close();
        steps_to_average++;
      }
  }
```

```
  char end_time_str[81];
  time_t end_time = time(NULL);
  tm *end_tmbuf=localtime(&end_time);
  int run_time = static_cast<int>(difftime(end_time,start_time));
  int hours = run_time/3600;
  int minutes = (run_time % 3600)/60;
  int seconds = (run_time % 3600)%60;

  //list<string>::iterator it;

  strftime(end_time_str, 80, "# Simulation successfully finished on %d
%B, %Y at %H:%M", end_tmbuf);
  cout << endl << end_time_str << endl;
  cout << "# Simulation took total of " << hours << " hours, " <<
minutes << " minutes and " << seconds << " seconds (" << run_time << "
seconds)" << endl;


  double NN = static_cast<double>(steps_to_average);
  avr_charge /= NN;
  avr_charge_sq /= NN;
  avr_q_plus /= NN;
  avr_q_plus_sq /= NN;
  avr_q_minus /= NN;
  avr_q_minus_sq /= NN;
  for (int site = 0; site < sys->get_size(); site++)
    avr_site_q[site] /= NN;

  cout << "Calculating correlation funciton..." << endl;

  for (int site = 0; site < sys->get_size(); site++)
    avr_correl[site] = avr_correl[site]/NN -
avr_site_q[0]*avr_site_q[site];



  double dr = 0.5*sys->get_lattice()->get_box()->Lx/
static_cast<double>(Nbin);
  double Lx_half = 0.5*sys->get_lattice()->get_box()->Lx, Ly_half =
sys->get_lattice()->get_box()->Ly;

  double* avr_gr_to_print = new double[Nbin];

  for(int bin = 0; bin < Nbin; bin++)
    avr_gr_to_print[bin] = 0.0;


  cout << "Binning..." << endl;

  for (int site_j = 0; site_j < sys->get_size(); site_j++)
```

```
  {
    double r = sys->get_lattice()->get_distance(0,site_j);
    if (r < Lx_half && r < Ly_half)
    {
       int bin = static_cast<int>(r/dr);
       if (bin < Nbin)
        avr_gr_to_print[bin] += avr_correl[site_j]/(M_PI*((bin+1)*(bin
+1) - bin*bin)*dr*dr);
    }
  }

  double charge_error = sqrt(avr_charge_sq - avr_charge*avr_charge)/
sqrt(NN-1);
  double charge_error_p = sqrt(avr_q_plus_sq - avr_q_plus*avr_q_plus)/
sqrt(NN-1);
  double charge_error_m = sqrt(avr_q_minus_sq -
avr_q_minus*avr_q_minus)/sqrt(NN-1);

  cout << endl <<
"------------------------------------------------------" << endl << endl;
  cout << "Average charge : " << avr_charge << " +/- " << charge_error
<< endl;
  if (single_component)
    if (is_acid)
      cout << "Dissociation rate : " << avr_charge/(-q[0]) << endl;
    else
      cout << "Dissociation rate : " << avr_charge/q[0] << endl;
  else
  {
    cout << "Average charge (positive) : " << avr_q_plus << " +/- " <<
charge_error_p << endl;
    cout << "Average charge (negative) : " << avr_q_minus << " +/- "
<< charge_error_m << endl;
    cout << "Dissociation rate (positive) : " << avr_q_plus/q[0] <<
endl;
    cout << "Dissociation rate (negative) : " << -avr_q_minus/q[1] <<
endl;
  }

  cout << endl <<
"------------------------------------------------------" << endl << endl;

  cout << "Printing averages..." << endl;
  ofstream avr_out((out_filename+"_averages.dat").c_str());
  if (!single_component)
    avr_out << pH << " " << avr_charge << " " << charge_error << " "
<< avr_q_plus << " " << charge_error_p << " " << avr_q_minus << " " <<
charge_error_m << " " << avr_q_plus/q[0] << " " << -avr_q_minus/q[1]
<< endl;
  else
```

```
    if (is_acid)
      avr_out << pH << " " << avr_charge << " " << charge_error << " "
<< avr_charge/(-q[0])  << endl;
    else
      avr_out << pH << " " << avr_charge << " " << charge_error << " "
<< avr_charge/q[0]  << endl;

  avr_out.close();

  cout << "Printing charges..." << endl;
  ofstream ch_out((out_filename+"_charges.dat").c_str());
  for (int i = 0; i < sys->get_size(); i++)
  {
    ch_out << sys->get_site(i)->x << " " << sys->get_site(i)->y << " "
<< ch_dist[i]/static_cast<double>(steps_to_average) << endl;
  }

  ch_out.close();

  cout << "Printing correlation..." << endl;
  ofstream out_gr((out_filename+"_pair_correl.dat").c_str());
  out_gr << "# bin   correl ";
  if (!single_component)
  {
    out_gr << " correl_pp  correl_mm  correl_pm " << endl;
  }
  else
    out_gr << endl;

  cout << "Printing 2d correl..." << endl;

  ofstream out_correl((out_filename+"_full_correl.dat").c_str());
  for (int site_j = 0; site_j < sys->get_size(); site_j++)
    out_correl << sys->get_site(site_j)->x << "  " << sys-
>get_site(site_j)->y << "  " << avr_correl[site_j]  << endl;

  out_correl.close();

  for(int bin = 0; bin < Nbin; bin++)
  {
    //out_gr << bin << "  " << avr_gr[bin]/NN-avr_charge*avr_charge <<
"  ";
    out_gr << bin << "  " << avr_gr_to_print[bin] << "  ";
    if (!single_component)
    {
      out_gr << avr_gr_pp[bin]/NN-avr_q_plus*avr_q_plus << "  " <<
avr_gr_mm[bin]/NN-avr_q_minus*avr_q_minus << "  " << avr_gr_pm[bin]/
NN-avr_q_plus*avr_q_minus << endl;
    }
    else
```

```
      out_gr << endl;
  }

  out_gr.close();



  //sys->get_lattice()->print_xyz(out_filename+"_charges.xyz");

  //sys->print_charges(out_filename+"_charges.dat",
static_cast<int>(L),static_cast<int>(L));
  //sys->print_energies(out_filename+"_energies.dat",
static_cast<int>(L), static_cast<int>(L));

//   delete mc;
//   delete sys;
//   delete [] gr;
//   delete [] gr_pp;
//   delete [] gr_mm;
//   delete [] gr_pm;
//   delete [] avr_gr;
//   delete [] avr_gr_pp;
//   delete [] avr_gr_mm;
//   delete [] avr_gr_pm;


  return 0;
}
```

```
#include "lattice.h"

double A1_X =  0.5;
double A1_Y = -0.5*sqrt(3.0);
double A2_X =  0.5;
double A2_Y =  0.5*sqrt(3.0);

static bool is_inside(double x, double y, Box& box)
{
  if (x < box.xlo) return false;
  if (x >= box.xhi) return false;
  if (y < box.ylo) return false;
  if (y >= box.yhi - 0.5*sqrt(3.0)) return false;
  return true;
}

Lattice::Lattice(double LX, double LY, double a) : m_box(LX,LY),
m_lattice_constant(a), m_has_type(false)
{
  double i, j;
  double x, y;
  int id;

  id = 0;
  j = 0.0;
  double y_min = 0.0;
  while (j < m_box.Ly)
  {
    i = 0.0;
    while (i < m_box.Ly)
    {
      x = i*m_lattice_constant*A1_X + j*m_lattice_constant*A2_X;
      y = i*m_lattice_constant*A1_Y + j*m_lattice_constant*A2_Y;
      if (is_inside(x,y,m_box))
      {
        m_lattice.push_back(new Site(id++,x,y,0.0,1,0.0));
        if (y < y_min) y_min = y;
      }
      if (j != 0.0)
      {
        x = i*m_lattice_constant*A1_X - j*m_lattice_constant*A2_X;
        y = i*m_lattice_constant*A1_Y - j*m_lattice_constant*A2_Y;
        if (is_inside(x,y,m_box))
        {
          m_lattice.push_back(new Site(id++,x,y,0.0,1,0.0));
          if (y < y_min) y_min = y;
        }
      }
      if (i != 0.0)
      {
```

```
        x = -i*m_lattice_constant*A1_X + j*m_lattice_constant*A2_X;
        y = -i*m_lattice_constant*A1_Y + j*m_lattice_constant*A2_Y;
        if (is_inside(x,y,m_box))
        {
          m_lattice.push_back(new Site(id++,x,y,0.0,1,0.0));
          if (y < y_min) y_min = y;
        }
        if (j != 0.0)
        {
          x = -i*m_lattice_constant*A1_X - j*m_lattice_constant*A2_X;
          y = -i*m_lattice_constant*A1_Y - j*m_lattice_constant*A2_Y;
          if (is_inside(x,y,m_box))
          {
            m_lattice.push_back(new Site(id++,x,y,0.0,1,0.0));
            if (y < y_min) y_min = y;
          }
        }
      }
      i += 1.0;
    }
    j += 1.0;
  }

  m_box.ylo = y_min;
  m_box.yhi = -y_min;
  m_box.Ly = m_box.yhi - m_box.ylo;

  m_size = m_lattice.size();

}

double Lattice::get_distance(int i, int j)
{
  Site* s1 = m_lattice[i];
  Site* s2 = m_lattice[j];

  double x1 = s1->x, y1 = s1->y, z1 = s1->z;
  double x2 = s2->x, y2 = s2->y, z2 = s2->z;

  double dx = x1 - x2, dy = y1 - y2, dz = z1 - z2;
  if (m_periodic)
  {
    if (dx > m_box.xhi) dx -= m_box.Lx;
    else if (dx < m_box.xlo) dx += m_box.Lx;
    if (dy > m_box.yhi) dy -= m_box.Ly;
    else if (dy < m_box.ylo) dy += m_box.Ly;
  }
  return sqrt(dx*dx + dy*dy + dz*dz);
}
```

```
void Lattice::build_neighbor_list(double kappa, double lb, double
r_cut)
{
  int i, j;
  Site* s1;
  Site* s2;
  double dist;


  m_eng_correction = lb*exp(-kappa*r_cut)/r_cut;

  for (i = 0; i < m_size; i++)
  {
    m_neighbors.push_back(Neighbors(i));
    for (j = 0; j < m_size; j++)
    {
      s2 = m_lattice[j];
      if (i != j)
      {
        dist = get_distance(i,j);
        if (dist <= r_cut)
        {
          double eng_fact = lb*exp(-kappa*dist)/dist;

m_neighbors[i].neighbors.push_back(Neighbor(s2,dist,eng_fact));
        }

      }
    }
    m_neighbors[i].n_neighbors = m_neighbors[i].neighbors.size();
  }
}

void Lattice::print_neighbors(int i)
{
  cout << "Neighbors of site " << i << " are : ";
  for (int j = 0; j < m_neighbors[i].n_neighbors; j++)
    cout << m_neighbors[i].neighbors[j].site->id << " " << endl;
  cout << endl;
}

void Lattice::print_xyz(string name)
{
  ofstream out;
  out.open(name.c_str());
  out << m_size << endl;
  out << "Some text" << endl;
  for (int i = 0; i < m_size; i++)
  {
    if (m_lattice[i]->type == 1)
```

```
      out << "C " << m_lattice[i]->x << " " << m_lattice[i]->y << " "
<< m_lattice[i]->z << endl;
    else
      out << "N " << m_lattice[i]->x << " " << m_lattice[i]->y << " "
<< m_lattice[i]->z << endl;
  }
  out.close();
}

void Lattice::swap_sites(int i, int j)
{
  double q = m_lattice[i]->q;
  double type = m_lattice[i]->type;
  m_lattice[i]->q = m_lattice[j]->q;
  m_lattice[i]->type = m_lattice[j]->type;
  m_lattice[j]->q = q;
  m_lattice[j]->type = type;
}
```

```
#ifndef __BOX_H__
#define __BOH_H__

struct Box
{
    Box(double LX, double LY, double LZ = 1.0)
    {
      Lx = LX;  Ly = LY; Lz = LZ;
      xlo = -0.5*LX; xhi = 0.5*LX;
      ylo = -0.5*LY; yhi = 0.5*LY;
      zlo = -0.5*LZ; zhi = 0.5*LZ;
    }
    double xlo, xhi;
    double ylo, yhi;
    double zlo, zhi;
    double Lx, Ly, Lz;
};

#endif
```

```
#  Monte Carlo simulation for solid/liquid membranes
#
#  Authors: Rastko Sknepnek (r-sknepnek@northwestern.edu)
#           Graziano Vernizzi (g-vernizzi@northwestern.edu)
#
#           Department of Materails Science and Engineering
#           Northwestern University
#
#  This program cannot be used, modified, and/or copied without
#  explicit permission of the authors.
#
#  (c) 2009, R. Sknepnek and G. Vernizzi
#

#################################
## Setup include directories and file lists for sub directories
include_directories($ENV{GSL_DIR}/include
                    ${MEMBRANE_SOURCE_DIR}/src/computes
                    ${MEMBRANE_SOURCE_DIR}/src/computes/vertex
                    ${MEMBRANE_SOURCE_DIR}/src/computes/edge
                    ${MEMBRANE_SOURCE_DIR}/src/computes/triangle
                    ${MEMBRANE_SOURCE_DIR}/src/cooling_curves
                    ${MEMBRANE_SOURCE_DIR}/src/driver
                    ${MEMBRANE_SOURCE_DIR}/src/driver/MC
                    ${MEMBRANE_SOURCE_DIR}/src/driver/FIRE
                    ${MEMBRANE_SOURCE_DIR}/src/driver/minimizer
                    ${MEMBRANE_SOURCE_DIR}/src/driver/BasinHopping
                    ${MEMBRANE_SOURCE_DIR}/src/dump
                    ${MEMBRANE_SOURCE_DIR}/src/log
                    ${MEMBRANE_SOURCE_DIR}/src/parser
                    ${MEMBRANE_SOURCE_DIR}/src/system
                    ${MEMBRANE_SOURCE_DIR}/src/utils
                    ${MEMBRANE_SOURCE_DIR}/src/constrainer
                    )

# list of all sources in various source directories
#file(GLOB MEMBRANE_SRCS ${MEMBRANE_SOURCE_DIR}/src/*.cpp $
{MEMBRANE_SOURCE_DIR}/src/*.h)
file(GLOB INITIAL_CONFIGURATION_BUILDER_SRC ${MEMBRANE_SOURCE_DIR}/
utils)

########################
## Configure the version info header file


# handle linux/mac and windows dates differently
if (NOT WIN32)
        exec_program("date" OUTPUT_VARIABLE COMPILE_DATE)
else(NOT WIN32)
        exec_program("cmd" ARGS "/c date /T" OUTPUT_VARIABLE
```

```
COMPILE_DATE)
endif (NOT WIN32)
```

```
#  Monte Carlo simulation for solid/liquid membranes
#
#  Authors: Rastko Sknepnek (r-sknepnek@northwestern.edu)
#           Graziano Vernizzi (g-vernizzi@northwestern.edu)
#
#           Department of Materails Science and Engineering
#           Northwestern University
#
#  This program cannot be used, modified, and/or copied without
#  explicit permission of the authors.
#
#  (c) 2009, R. Sknepnek and G. Vernizzi
#

# first, see if we can get any supported version of Math Kernel
Library
if(USE_MKL_LIBRARY)
  find_path(MKL_INCLUDE_PATH mkl_vsl.h /opt/intel/
composerxe-2011.4.191/mkl/include /opt/intel/composerxe/mkl/include /
opt/intel/Compiler/11.1/046/mkl/include/ /opt/intel/compilers/11/mkl/
include/ /opt/intel/mkl/include)
  find_path(MKL_LIB_PATH libmkl_core.a /opt/intel/
composerxe-2011.4.191/mkl/lib/intel64 /opt/intel/composerxe/mkl/lib/
intel64 /opt/intel/Compiler/11.1/046/mkl/lib/em64t /opt/intel/
compilers/11/mkl/lib/em64t /opt/intel/mkl/lib/intel64)

  #find_path(MKL_IOMP5_PATH libiomp5.a /opt/intel/
composerxe-2011.4.191/compiler/lib/intel64/ /opt/intel/Compiler/
11.1/046/lib/intel64 /opt/intel/compilers/11/lib/intel64/)

  include_directories(${MKL_INCLUDE_PATH})
  #link_directories(${MKL_LIB_PATH} ${MKL_IOMP5_PATH})
  link_directories(${MKL_LIB_PATH})

  set(MKL_LIB_SEARCHPATH /opt/intel/composerxe-2011.4.191/mkl/lib/
intel64 /opt/intel/composerxe/mkl/lib/intel64 /opt/intel/Compiler/
11.1/046/mkl/lib/em64t /opt/intel/compilers/11/mkl/lib/em64t/ /opt/
intel/mkl/lib/intel64)
  #set(MKL_IOMP5_SEARCHPATH /opt/intel/Compiler/11.1/046/lib/intel64 /
opt/intel/compilers/11/lib/intel64/)

  set(MKL_LIBS mkl_intel_lp64 mkl_sequential mkl_core)
  #set(MKL_IOMP5 iomp5)

  foreach ( LIB ${MKL_LIBS})
          find_library(${LIB}_PATH ${LIB} ${MKL_LIB_SEARCHPATH})
  endforeach ( LIB )

  #foreach ( LIB ${MKL_IOMP5})
  #        find_library(${LIB}_PATH ${LIB} ${MKL_IOMP5_SEARCHPATH})
```

```
  #endforeach ( LIB )
endif(USE_MKL_LIBRARY)
```

```
#  Monte Carlo simulation for solid/liquid membranes
#
#  Authors: Rastko Sknepnek (r-sknepnek@northwestern.edu)
#           Graziano Vernizzi (g-vernizzi@northwestern.edu)
#
#           Department of Materails Science and Engineering
#           Northwestern University
#
#  This program cannot be used, modified, and/or copied without
#  explicit permission of the authors.
#
#  (c) 2009, R. Sknepnek and G. Vernizzi
#

CMAKE_MINIMUM_REQUIRED(VERSION 2.6 FATAL_ERROR)
if(COMMAND cmake_policy)
        cmake_policy(SET CMP0003 NEW)
endif(COMMAND cmake_policy)


project (MEMBRANE)


if(ENABLE_STATIC)
  SET (CMAKE_EXE_LINKER_FLAGS "${CMAKE_EXE_LINKER_FLAGS} -static")
  SET (THREAD_LIB -lpthread)
endif(ENABLE_STATIC)

option(USE_CGAL_LIBRARY "Use CGAL library with building initialization
tools (has problems with static builds" ON)

# Setup a number of misc options and libraries
include (CMakeMiscSetup.txt)
# Find the boost libraries and set them up
include (CMakeBoostSetup.txt)
# Find GSL libraries
include (CMakeGSLSetup.txt)
# Set default CFlags
include (CMakeCFlagsSetup.txt)
# Configure some source files, include directories, and create
variables listing all source files
include (CMakeSRCSetup.txt)
# Include GiNaC libray
include (CMakeGiNaCSetup.txt)

################################
## Define common libraries used by every target in MEMBRANE
set(BOOST_LIBS  ${Boost_PROGRAM_OPTIONS_LIBRARY} $
{Boost_REGEX_LIBRARY} )
set(GSL_LIBS ${GSL_LIBRARIES})
```

```
set(MATH_LIB -lm)
set(MEMBRANE_LIBS ${BOOST_LIBS} ${GSL_LIBS} ${MATH_LIB})

# ################################################
# place all executables in the build directory
set (CMAKE_RUNTIME_OUTPUT_DIRECTORY ${CMAKE_BINARY_DIR})

################################
## include documentation directories
if (ENABLE_DOXYGEN)
        add_subdirectory (doc)
endif (ENABLE_DOXYGEN)

add_subdirectory(src)

add_executable(QET_build ${INITIAL_CONFIGURATION_BUILDER_SRC}/QET_V2)
target_link_libraries(QET_build ${MATH_LIB})
add_executable(QET_fast_build ${INITIAL_CONFIGURATION_BUILDER_SRC}/
QET_V3)
target_link_libraries(QET_fast_build ${MATH_LIB})
add_executable(QET_icosa ${INITIAL_CONFIGURATION_BUILDER_SRC}/
QET_icosa)
target_link_libraries(QET_icosa ${MATH_LIB})
add_executable(QET_charge ${INITIAL_CONFIGURATION_BUILDER_SRC}/
QET_electrostatics)
target_link_libraries(QET_charge ${MATH_LIB})
add_executable(QET_stoich ${INITIAL_CONFIGURATION_BUILDER_SRC}/
StoichiometricTNumbers)
target_link_libraries(QET_stoich ${MATH_LIB})
add_executable(QET_flat ${INITIAL_CONFIGURATION_BUILDER_SRC}/flat)
target_link_libraries(QET_flat ${MATH_LIB})
add_executable(QET_pentagon ${INITIAL_CONFIGURATION_BUILDER_SRC}/
flatPentagon)
target_link_libraries(QET_pentagon ${MATH_LIB})

# If enabled, include CGAL library
if (USE_CGAL_LIBRARY)
 include (CMakeCGALSetup.txt)
endif (USE_CGAL_LIBRARY)

if (FOUND_CGAL_LIBRARY)
  link_libraries(${BOOST_LIBS} ${MEMBRANE_LIBS})
  add_executable(Random_mesh ${INITIAL_CONFIGURATION_BUILDER_SRC}/
random_mesh)
  add_executable(triangulate_disk $
{INITIAL_CONFIGURATION_BUILDER_SRC}/triangulate_disk)
endif (FOUND_CGAL_LIBRARY)
```

```
#  Monte Carlo simulation for solid/liquid membranes
#
#  Authors: Rastko Sknepnek (r-sknepnek@northwestern.edu)
#           Graziano Vernizzi (g-vernizzi@northwestern.edu)
#
#           Department of Materails Science and Engineering
#           Northwestern University
#
#  This program cannot be used, modified, and/or copied without
#  explicit permission of the authors.
#
#  (c) 2009, R. Sknepnek and G. Vernizzi
#

#find_package(GSL REQUIRED)
#include_directories(${GSL_INCLUDE_DIRS})

#MESSAGE("Searching for GSL lib")

FIND_LIBRARY(GSL_LIB_PATH gsl)
FIND_PATH(GSL_INCLUDE_PATH gsl/gsl_rng.h )
  IF ( GSL_LIB_PATH AND GSL_INCLUDE_PATH )
    #MESSAGE("-- Looking for Ginac - found")
    SET( GSL_INCLUDE "${GSL_INCLUDE_PATH}")
    SET( GSL_LIBRARIES "-lgsl -lgslcblas")
    include_directories(${GSL_INCLUDE_PATH})
    link_directories(${GSL_LIB_PATH})
    #MESSAGE( "-- Ginac includes ${GINAC_INCLUDE}")
    #MESSAGE( "-- Ginac libs     ${GINAC_LIBS}")
  ELSE ( GSL_LIB_PATH AND GSL_INCLUDE_PATH )
    MESSAGE(FATAL_ERROR "GSL not found")
  ENDIF ( GSL_LIB_PATH AND GSL_INCLUDE_PATH )
```

```
#
#  Simulator of solid/liquid membranes.
#
#  Author: Rastko Sknepnek (sknepnek@gmail.com)
#
#  Department of Materials Sciences and Engineering
#  Northwestern University
#
#  (c) 2009, 2010, 2011, 2012, all rights reserved
#
#  Department of Physics
#  Syracuse University
#
#  (c) 2012, all rights reserved
#
#  This program cannot be used, copied, or modified without
#  explicit permission of the author.
#
# ############################################################
#
#  Try to locate GiNaC library
#
# ############################################################

OPTION( GINAC_SUPPORT "Turn me off to disable Ginac support" OFF )

SET(GINAC_FOUND FALSE)

IF (GINAC_SUPPORT)

  #MESSAGE("Searching for ginac lib")

  FIND_LIBRARY(GINAC_LIB_PATH ginac)
  FIND_PATH(GINAC_INCLUDE_PATH ginac/ginac.h )
  IF ( GINAC_LIB_PATH AND GINAC_INCLUDE_PATH )
    #MESSAGE("-- Looking for Ginac - found")
    SET( GINAC_INCLUDE "${GINAC_INCLUDE_PATH}")
    SET( GINAC_LIBS "${GINAC_LIB_PATH}")
    #MESSAGE( "-- Ginac includes ${GINAC_INCLUDE}")
    #MESSAGE( "-- Ginac libs     ${GINAC_LIBS}")
    ADD_DEFINITIONS(-DENABLE_GINAC)
    SET (GINAC_FOUND TRUE)
  ENDIF ( GINAC_LIB_PATH AND GINAC_INCLUDE_PATH )

ENDIF (GINAC_SUPPORT)
```

```
#  Monte Carlo simulation for solid/liquid membranes
#
#  Authors: Rastko Sknepnek (r-sknepnek@northwestern.edu)
#           Graziano Vernizzi (g-vernizzi@northwestern.edu)
#
#           Department of Materails Science and Engineering
#           Northwestern University
#
#  This program cannot be used, modified, and/or copied without
#  explicit permission of the authors.
#
#  (c) 2009, 2010, 2011 R. Sknepnek and G. Vernizzi
#

set(CMAKE_ALLOW_LOOSE_LOOP_CONSTRUCTS true)

if ( COMMAND cmake_policy )
  cmake_policy( SET CMP0003 NEW )
endif()

if(POLICY CMP0011)
   cmake_policy(SET CMP0011 NEW) # or even better, NEW
endif(POLICY CMP0011)

# setup the CGAL static linkage
if(ENABLE_STATIC)
set(CGAL_BUILD_SHARED_LIBS "OFF")
else(ENABLE_STATIC)
set(CGAL_BUILD_SHARED_LIBS "ON")
endif(ENABLE_STATIC)


find_package(CGAL  COMPONENTS Core )

if ( CGAL_FOUND )

  include( ${CGAL_USE_FILE} )

  include( CGAL_CreateSingleSourceCGALProgram )

  set(FOUND_CGAL_LIBRARY true)

else()

  set(FOUND_CGAL_LIBRARY false)

endif()
```

```
#  Monte Carlo simulation for solid/liquid membranes
#
#  Authors: Rastko Sknepnek (r-sknepnek@northwestern.edu)
#           Graziano Vernizzi (g-vernizzi@northwestern.edu)
#
#           Department of Materails Science and Engineering
#           Northwestern University
#
#  This program cannot be used, modified, and/or copied without
#  explicit permission of the authors.
#
#  (c) 2009, R. Sknepnek and G. Vernizzi
#


#################################
## Setup default CXXFLAGS
if(NOT PASSED_FIRST_CONFIGURE)
        message(STATUS "Overriding CMake's default CFLAGS (this should
appear only once)")

        if(CMAKE_COMPILER_IS_GNUCXX)
                # special handling to honor gentoo flags
                if (HONOR_GENTOO_FLAGS)
                set(CMAKE_CXX_FLAGS_DEBUG "-Wall" CACHE STRING "Flags
used by the compiler during debug builds." FORCE)
                set(CMAKE_CXX_FLAGS_MINSIZEREL "-Wall -DNDEBUG" CACHE
STRING "Flags used by the compiler during minimum size release
builds." FORCE)
                set(CMAKE_CXX_FLAGS_RELEASE "-DNDEBUG -Wall" CACHE
STRING "Flags used by the compiler during release builds." FORCE)
                set(CMAKE_CXX_FLAGS_RELWITHDEBINFO "-DNDEBUG -Wall"
CACHE STRING "Flags used by the compiler during release builds with
debug info." FORCE)

                else (HONOR_GENTOO_FLAGS)

                # default flags for g++
                # default build type is Release when compiling make
files
                if(NOT CMAKE_BUILD_TYPE)
                        if(${CMAKE_GENERATOR} STREQUAL "Xcode")
                        else(${CMAKE_GENERATOR} STREQUAL "Xcode")
                                set(CMAKE_BUILD_TYPE "Release" CACHE
STRING  "Build type: options are None, Release, Debug, RelWithDebInfo"
FORCE)
                        endif(${CMAKE_GENERATOR} STREQUAL "Xcode")
                endif(NOT CMAKE_BUILD_TYPE)

                set(CMAKE_CXX_FLAGS_DEBUG "-g -Wall" CACHE STRING
```

```
"Flags used by the compiler during debug builds." FORCE)
                set(CMAKE_CXX_FLAGS_MINSIZEREL "-Os -Wall -DNDEBUG"
CACHE STRING "Flags used by the compiler during minimum size release
builds." FORCE)
                set(CMAKE_CXX_FLAGS_RELEASE "-O3 -funroll-loops -
ffast-math -DNDEBUG -Wall" CACHE STRING "Flags used by the compiler
during release builds." FORCE)
                set(CMAKE_CXX_FLAGS_RELWITHDEBINFO "-g -O -funroll-
loops -ffast-math -DNDEBUG -Wall" CACHE STRING "Flags used by the
compiler during release builds with debug info." FORCE)

                endif (HONOR_GENTOO_FLAGS)
        elseif(MSVC80)
                # default flags for visual studio 2005
                if (CMAKE_CL_64)
                        set(SSE_OPT "")
                else (CMAKE_CL_64)
                        set(SSE_OPT "/arch:SSE2")
                endif (CMAKE_CL_64)

                set(CMAKE_CXX_FLAGS "/DWIN32 /D_WINDOWS /W3 /Zm1000 /
EHa /GR /MP" CACHE STRING "Flags used by all build types." FORCE)
                set(CMAKE_CXX_FLAGS_RELEASE "${SSE_OPT} /Oi /Ot /Oy /
fp:fast /MD /Ox /Ob2 /D NDEBUG" CACHE STRING "Flags used by the
compiler during release builds." FORCE)
                set(CMAKE_CXX_FLAGS_RELWITHDEBINFO "${SSE_OPT} /Oi /
Ot /Oy /fp:fast /MD /Zi /Ox /Ob1 /D NDEBUG" CACHE STRING "Flags used
by the compiler during release builds with debug info." FORCE)
                set(CMAKE_CXX_FLAGS_MINSIZEREL "${SSE_OPT} /Oi /Ot /
Oy /fp:fast /MD /O1 /Ob1 /D NDEBUG" CACHE STRING "Flags used by the
compiler during minimum size release builds." FORCE)

        else(CMAKE_COMPILER_IS_GNUCXX)
                message(SEND_ERROR "No default CXXFLAGS for your
compiler, set them manually")
        endif(CMAKE_COMPILER_IS_GNUCXX)

SET(PASSED_FIRST_CONFIGURE ON CACHE INTERNAL "First configure has run:
CXX_FLAGS have had their defaults changed" FORCE)
endif(NOT PASSED_FIRST_CONFIGURE)
```

```
#  Monte Carlo simulation for solid/liquid membranes
#
#  Authors: Rastko Sknepnek (r-sknepnek@northwestern.edu)
#           Graziano Vernizzi (g-vernizzi@northwestern.edu)
#
#           Department of Materails Science and Engineering
#           Northwestern University
#
#  This program cannot be used, modified, and/or copied without
#  explicit permission of the authors.
#
#  (c) 2009, R. Sknepnek and G. Vernizzi
#

#################################
## Boost is a required library

# setup the boost static linkage
if(ENABLE_STATIC)
set(Boost_USE_STATIC_LIBS "ON")
else(ENABLE_STATIC)
set(Boost_USE_STATIC_LIBS "OFF")
endif(ENABLE_STATIC)

set(Boost_USE_MULTITHREAD "OFF")

# setup some additional boost versions so that the newest versions of
boost will be found
#set(Boost_ADDITIONAL_VERSIONS "1.41.0;1.41;1.40.0;1.40;1.39.0;1.39")
set(Boost_ADDITIONAL_VERSIONS "1.55.0" "1.55" "1.54.0" "1.54" "1.53"
"1.52" "1.51" "1.51.0" "1.50" "1.50.0" "1.49" "1.49.0" "1.48" "1.48.0"
"1.48.0.2" "1.47" "1.47.0" "1.46.1" "1.46" "1.46.0" "1.45" "1.45.0"
"1.44" "1.44.0" "1.42" "1.42.0" "1.41.0" "1.41" "1.40.0" "1.40"
"1.39.0" "1.39" "1.38.0" "1.38" "1.37.0" "1.37" )

# first, see if we can get any supported version of Boost
#find_package(Boost 1.48 COMPONENTS program_options regex REQUIRED)
#find_package(Boost COMPONENTS program_options regex REQUIRED)
find_package(Boost COMPONENTS program_options REQUIRED)

# if we get boost 1.35 or greater, we need to get the system library
too

#if (Boost_MINOR_VERSION GREATER 34)
#find_package(Boost 1.48.0 COMPONENTS program_options regex REQUIRED)
#endif (Boost_MINOR_VERSION GREATER 34)

include_directories(${Boost_INCLUDE_DIR})
```

Codes for the paper “Polydispersity-driven topological defects as order-restoring excitations”

======= cluster.cpp

```
//////////////////////////////////////////////////
////soft particles with dispersity/////////////////
////by Zhenwei Yao ////////////////////////////////
////Date: June 24, 2013////////////////////////////
//////////////////////////////////////////////////

#include "variables.h"

double Vij (int arg_i, int arg_j);
double Vi (int arg_i);
vec move_s (vec arg_r, vec arg_s);
void do_MC ();

//////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////
/////main function//////////////////////////////////////
//////////////////////////////////////////////////////////

int main(){
	double beta = 1.9;
	int Nsw = 1000000; //1000000

	int nL = 10;
	diameter_avg = R/nL;
	s = 0.01*diameter_avg;

	double Tini;
	Temp = pow(1-0.5,alpha)*0.0000001;
	Tini = Temp;

	cout<<"initial temperature is: "<<Temp<<endl;
	cout<<"diameter_avg = "<<diameter_avg<< " ; beta = " <<beta<<endl;
	cout<<"alpha = "<<alpha<<"; N="<<N<<" ; nL = "<<nL<<endl;

	////specify diameter///
	//////////////////////
```

```
int i, j, m, l;
for (i=1; i<=N; i++){
  diameter[i] = diameter_avg*1.0;
}


////////specify initial positions of disks////////
////////////////////////////////////////////////
double energy[Nsw+1];
vec v_temp;
double pro;
/* initialize random seed: */
srand (time(NULL));

double a = diameter_avg;
i = 0;
for (l=0; l<=nL; l++){
  for(m = 0; m <= 2*nL-abs(l); m++){
  i++;
  v_temp.set_x(-nL*a+a/2.0*l+m*a); //such that the last item at level l has x-
coordinates: na-al/2
  v_temp.set_y(sqrt(3.0)/2.0*a*l);
  pt[i].set_X(v_temp);
  //pt_new[i].set_X(v_temp);
  }}

diameter[nL+1] = diameter_avg * beta;

for (l=-1; l>=-nL; l=l-1){
  for(m = 0; m <= 2*nL-abs(l); m++){
  i++;
  v_temp.set_x(-nL*a+a/2.0*abs(l)+m*a);
  v_temp.set_y(-sqrt(3.0)/2.0*a*abs(l));
  pt[i].set_X(v_temp);
  //pt_new[i].set_X(v_temp);
  }}

///////save initial positions////////
////////////////////////////////////
ofstream coordinates_initial_ifile;
coordinates_initial_ifile.open("output_files/points_coordinates_initial.txt");
for(i=1;i<=N;i++){
```

```
coordinates_initial_ifile<<pt[i].get_X().get_x()<<"\t"<<pt[i].get_X().get_y()<<endl;
        }
        coordinates_initial_ifile.close();

        /////energy of initial state////
        ////////////////////////////////
        energy_ini = 0;
        for(i = 1; i <= N; i++){
            energy_ini = energy_ini + Vi(i);
        }
        //energy_ini = energy_ini/2.0;
        energy[0] = energy_ini;
        cout<<"the energy of initial state is: "<<energy[0]<<endl;


        ///////////////
        //do MC////////
        ///////////////

        double dT = 0.9999;
        for (j = 1; j <= Nsw; j++){
          do_MC(); //changes the global variables: pt[N+1], energy_ini
          energy[j] = energy_ini;
          if ( abs(energy[j] - energy[j-1])/energy[j-1]  < 0.0000001) {
            Temp = dT*Temp;
            //cout<<" temperature decreased to "<<Temp<<endl;
          }
          if (j%(Nsw/10) == 0)
            cout<<"ns = "<<j<<" Temperature = "<<Temp<<" ; s = diameter_avg *
"<<s/diameter_avg<<" energy is "<<energy[j]<<endl;
        }

        ////////////////////////////////
        ///write to files/////////////////
        ////////////////////////////////

        ofstream parameter_ifile;

        parameter_ifile.open("output_files/parameters.txt");
        parameter_ifile<<"step length: s = "<<s<<" ; R = "<<R<<" ; N = "<<N<<"
; alpha = "<<alpha<<"; diameter ="<<diameter_avg \
```

```
          <<" ; beta = "<<beta<<" ; Tinitial = "<<Tini<< "; T = dT * T, with dT
="<< dT<< " ; total sweeps Nsw = "<<Nsw<< endl;
        parameter_ifile.close();

        ofstream coordinates_ifile, energy_ifile, energy_i_ifile;

        coordinates_ifile.open("output_files/points_coordinates.txt");
        for(i=1;i<=N;i++){
         coordinates_ifile<<pt[i].get_X().get_x()<<"\t"<<pt[i].get_X().get_y()<<endl;
        }
        coordinates_ifile.close();

        energy_ifile.open("output_files/tot_energy.txt");
        for(i=1;i<=Nsw;i++){
         if (i%(Nsw/50) == 0)
         energy_ifile<<i<<"\t"<<energy[i]<<endl; // 50 records in total
        }
        energy_ifile.close();

        energy_i_ifile.open("output_files/energy_i.txt");
        for(i=1;i<=N;i++){
         energy_i_ifile<<i<<"\t"<<Vi(i)<<endl; //
        }
        energy_i_ifile.close();

        return 0;
}


========== do_MC.cpp
#include "variables.h"
vec move_s (vec arg_r, vec arg_s);
double Vi (int arg_i);

void do_MC(){   //changes the global variables: pt[N+1], energy_ini, denergy.
Gamma is also a global constant
        double beta, beta_r, energy_new, pro, pro_theta;
        double Viold, Vinew;
        vec r_old;
        int i,j;
        vec vec_s, r_new, vec_t, ri_hat;
        for(i = 1; i <= N; i++){ //, i!= ((nL+1)/2)*nL + ((nL+1)/2)
         if (i == nL+1 ){  //fix the middle disk
           continue;
```

```
        }

        r_old = pt[i].get_X();
        Viold = Vi(i);
        //move a step
        srand (time(NULL));
        pro_theta = rand()%100000;
        pro_theta = pro_theta/100000;  // in [0,1]
        beta = pro_theta*2*Pi; // in [0, 2*Pi]

        /*
        srand (time(NULL));
        pro_s = rand()%100000;
        pro_s = pro_s/100000 -0.5;  // in [0,1]-0.5 = [-0.5,0.5]
        amp = pro_s*2.0*s; // in [-s, s]
*/
        vec_s.set_x(s*cos(beta));
        vec_s.set_y(s*sin(beta));
        r_new = move_s(pt[i].get_X(),vec_s);

        if ( r_new.get_x() < -R || r_new.get_x() > R \
           ||  r_new.get_y() < -sqrt(3.0)/2.0*R ||  r_new.get_y() > sqrt(3.0)/2.0*R \
           || r_new.get_y() < -sqrt(3.0)* (R- abs(r_new.get_x()))  || r_new.get_y() >
sqrt(3.0)* (R- abs(r_new.get_x())) ) {  //i.e., r_new is outside the hexagonal disk
          r_new = pt[i].get_X();
          continue;
        }

        pt[i].set_X(r_new); //change pt[i]
        Vinew = Vi(i);
        //calculate the change of energy
        denergy = Vinew - Viold;

        pt[i].set_X(r_old); //reset pt[i]
        //judge if accept the step
        if (denergy < 0) { pt[i].set_X(r_new); energy_ini = energy_ini + denergy;
continue; }
        if (denergy/Temp > 75) continue;
        else {
        srand (time(NULL));
        pro = rand()%100000;
        pro = pro/100000;
        if ( pro < exp(-denergy/Temp) ){
              pt[i].set_X(r_new);
```

```
                energy_ini = energy_ini + denergy;
                continue;
               }
               else continue;
          }

        } //end of for
};

========== Point.h
#ifndef POINT_H
#define POINT_H

#include "variables.h"

class Point{
        vec X, s_vec;  //position of a point is X and its associated s_vec generate a
new point: X+s_vec.
public:
        ////////////////////
        //specify values////
        ////////////////////


        ////////////////////////
        ///set member data/////
        ////////////////////////
        void set_X(vec other){
          X = other;
        };
        void set_s_vec(vec other){
          s_vec = other;
        };


        ////////////////////////
        ///visit member data///
        ////////////////////////

        vec get_X(){
          return X;
        };
        vec get_s_vec(){
```

```
        return s_vec;
      };

};



#endif
```

Codes for the paper “Electrostatic repulsion-driven crystallization model arising from filament networks”

```
======== bessko.cpp
#include "variables.h"
double bessi0(double x);


double bessk0(double x)  //Ref: Numerical Recipes in C, The Art of Scientific Computing Second Edition
{
        //double bessi0(double x);
        double y,ans;

        if (x <= 2.0) {
          y=x*x/4.0;
          ans=(-log(x/2.0)*bessi0(x))+(-0.57721566+y*(0.42278420
                +y*(0.23069756+y*(0.3488590e-1+y*(0.262698e-2
                +y*(0.10750e-3+y*0.74e-5))))));
        } else {
          y=2.0/x;
          ans=(exp(-x)/sqrt(x))*(1.25331414+y*(-0.7832358e-1
                +y*(0.2189568e-1+y*(-0.1062446e-1+y*(0.587872e-2
                +y*(-0.251540e-2+y*0.53208e-3))))));
        }
        return ans;
};


========= do_MC.cpp
#include "variables.h"
vec move_s (vec arg_r, vec arg_s);
double Vi (vec arg_vi, int arg_i);

void do_MC(){   //changes the global variables: pt[N+1], energy_ini, denergy. Gamma is also a global constant
        double beta, beta_r, energy_new, pro;
        int i,j;
        vec vec_s, r_new, vec_t, ri_hat;
        for(i = 1; i <= N; i++){
          //move a step
          pro = rand()%100000;
          pro = pro/100000;  // in [0,1]
          beta = pro*2*Pi; // in [0, 2*Pi]
```

```
            vec_s.set_x(s*cos(beta));
            vec_s.set_y(s*sin(beta));
            r_new = move_s(pt[i].get_X(),vec_s);
            if (r_new.norm() > R) {
              r_new = move_s(pt[i].get_X(),vec_s.inverse());

              pro = rand()%100000;
              pro = pro/100000;  // in [0,1]
              beta_r = pro*(Pi-s/R) + Pi/2+s/(2*R); // in [Pi/2 + s/(2R), 3Pi/2 - s/(2R)]

              vec_t.set_x(-1.0*pt[i].get_X().get_y()/pt[i].get_X().norm());
              vec_t.set_y(pt[i].get_X().get_x()/pt[i].get_X().norm());
              ri_hat.set_x(pt[i].get_X().get_x()/pt[i].get_X().norm());
              ri_hat.set_y(pt[i].get_X().get_y()/pt[i].get_X().norm());

              vec_s.set_x(s*cos(beta_r)*ri_hat.get_x() + s*sin(beta_r)*vec_t.get_x()  );
              vec_s.set_y(s*cos(beta_r)*ri_hat.get_y() + s*sin(beta_r)*vec_t.get_y()  );

              r_new = move_s(pt[i].get_X(),vec_s);
            }
            //calculate the change of energy
            denergy = Omega*pow(r_new.norm()/R,2.0)*exp(kappa*(r_new.norm()-R)) -
Omega*pow(pt[i].get_X().norm()/R,2.0)*exp(kappa*(pt[i].get_X().norm()-R))+
Vi(r_new, i) - Vi(pt[i].get_X(),i);

           // denergy = Gamma*(pow(r_new.norm(),2) - pow(pt[i].get_X().norm(),2) ) +
Vi(r_new, i) - Vi(pt[i].get_X(),i);
            //energy_ini = energy_ini + denergy;

            //judge if accept the step
            if (denergy < 0) { pt[i].set_X(r_new); energy_ini = energy_ini + denergy;
continue; }
            if (denergy/energy_ini > 75) continue;
                 else {
                 pro = rand()%100000;
                 pro = pro/100000;
                 if ( pro < exp(-denergy/Temp) ){
                   pt[i].set_X(r_new);
                   energy_ini = energy_ini + denergy;
                   continue;
                 }
                 else continue;
            }
```

```
        } //end of for

};

======= move_s.cpp
#include "variables.h"

vec move_s (vec arg_r, vec arg_s){
        vec arg_rp;
        arg_rp.set_x(arg_r.get_x()+arg_s.get_x());
        arg_rp.set_y(arg_r.get_y()+arg_s.get_y());
        return arg_rp;
};
```

Codes for the paper “Packing of charged chains on toroidal geometries”

========== chain.cpp

```
////////////////////////////////////////////////
////ground state shape of a chain on torus////////
////by Zhenwei Yao ////////////////////////////
////Date: June 8, 2012/////////////////////////
////////////////////////////////////////////////

#include "header.h"
#include "vec.h"
//#include "test.h"
#include "Point.h"
#include <time.h>
#include <fstream>


double get_Vij (vec arg1, vec arg2){
	double arg_d;
	vec diff;
	diff =  arg1 - arg2;
	arg_d = (1/diff.norm())*exp(-(diff.norm())/lambda);   // *exp(-
(diff.norm())/lambda)
	return arg_d;
}


// with given alpha of pt[0] and all beta[N], we can specify all points (X_A, p_normal
and calibrated X)
// this function returns all points
//theta of the first point is DEFINED to be unity
Point * specify_points(double alpha_initial_arg, double beta_arg[]){
	int i;
	Point pt_temp[N + 1];
	////speficy pt_temp[0]
	vec pt_first(R1+R2*cos(alpha_initial_arg),0,R2*sin(alpha_initial_arg));
//alpha_initial_arg determines the first point
	pt_temp[0].set_X(pt_first); //specify X for pt[0]
	pt_temp[0].set_XA_N_X();   //speficy X_A, p_normal and the calibrated X for
pt[0]
	pt_temp[0].set_beta(beta_arg[0]); //speficy beta associated with pt[0]
```

```
        ////specify ealpha
        vec ealpha(-d12*sin(alpha_initial_arg), 0, d12*cos(alpha_initial_arg)); //get the
ealpha, whose length is d12
        ///to find pt[1]
        vec v01; //v01 = X1 - X0
        v01 = ealpha.rotate(pt_temp[0].get_p_normal(), beta_arg[0]);
        pt_temp[1].set_X(pt_temp[0].get_X() + v01);
        pt_temp[1].set_XA_N_X();   //speficy X_A, p_normal and the calibrated X for
pt[1]
        pt_temp[1].set_beta(beta_arg[1]); //speficy beta associated with pt[1]
        //set all points from pt[2] to pt[N]
        for(i = 2; i <= N; i++){
          pt_temp[i].set_X(  pt_temp[i-1].get_X() \
               + ((pt_temp[i-2].get_X()-pt_temp[i-1].get_X())/((pt_temp[i-2].get_X()-
pt_temp[i-1].get_X()).norm())*d12 ).rotate(pt_temp[i-1].get_p_normal(),beta_arg[i-
1])  );
           //rotate v(i-2 to i-1), and get pt[i]
           pt_temp[i].set_XA_N_X(); //calibrate X of i
           pt_temp[i].set_beta(beta_arg[i]); // set the angle note: beta_arg[N] is not
specified
        };
        return pt_temp;
}

//calculate energy for a given configuration
double energy(Point pt_arg[], int N_arg){   //N_arg: energy of pt[0]...pt[N_arg]
        double tv = 0;
        int i, j;
        for(i = 0; i <= N_arg-1; i++){
          for (j = i+1; j<=N_arg;j++){
               tv = tv + get_Vij(pt_arg[i].get_X(), pt_arg[j].get_X());
               }
          } // end of the two for's
        return tv;
}



////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
/////main function//////////////////////////////////////
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
```

```
int main(){
        int i, j;
        int m_RW, M_RW = 1000; //number of RWs
        double d12; //bond distance
        double beta[N]; //rotating angle from pt 0 to N-1
        Point pt[N+1];
        double pro;

        cout<<"initial condition: pi/2, pi/2; N = 200; M_RW = 1000; R1 = 1.2
"<<endl;

        //to initialize pt[N]
        beta[0] = Pi/2;
        for (i=1; i<= N/5; i++){
          beta[i] =Pi+0.01;
        };
        for (i=N/5; i<= 2*N/5; i++){
          beta[i] =Pi+0.05;
        };
        for (i=2*N/5; i<= 3*N/5; i++){
          beta[i] =Pi-0.05;
        };
        for (i=3*N/5; i<= N-1; i++){
          beta[i] =Pi-0.1;
        };
        for (i=0; i<=N; i++){
          pt[i] = specify_points(alpha_i, beta)[i];
          //pt_ini[i] = specify_points(alpha_i, beta)[i];
        };

        //generate a random chain
        beta[0] = beta_i;
        pt[0] = specify_points(alpha_i, beta)[0];
        pt[1] = specify_points(alpha_i, beta)[1];
        double em = 0.0;


        ofstream ek_file;
        ek_file.open("output_files/ek.txt",ios::app | ios::ate);
        for (lambda = 0.1; lambda <= 1.5; lambda = lambda + 0.02){
          em = 0.0;
```

```
        for (m_RW = 1; m_RW <= M_RW; m_RW ++){
          for(i=1; i<=N-1; i++){
          //determined position of pt[2]
          pro = rand()%100000;  //rand()%10: return a random number between 0
and 9
          pro = pro/100000;
               beta[i] = pro*2*Pi;
          pt[i+1] = specify_points(alpha_i, beta)[i+1];
          }
          em = em + energy(pt,N);
          }
        ek_file<<lambda<<"\t"<<em/M_RW<<endl;
        if (lambda > 0.5 & lambda <0.53)  cout<<"half finished "<<endl;
        }
        ek_file.close();


   return 0;
}


========== vec.cpp

#include "vec.h"

vec::vec(double arg_1, double arg_2, double arg_3){ //to initialize a created vec: vec
instance(1,2,3);
          x_component =  arg_1;
          y_component =  arg_2;
          z_component =  arg_3;
        };
double vec::get_x(){
          return x_component;
        };
double vec::get_y(){
          return y_component;
        };
double vec::get_z(){
          return z_component;
        };


vec vec::operator + (vec arg){
          vec temp;
```

```
        temp.x_component = x_component + arg.x_component;
        temp.y_component = y_component + arg.y_component;
        temp.z_component = z_component + arg.z_component;
        return(temp);
       };
vec vec::operator - (vec arg){
        vec temp;
        temp.x_component = x_component - arg.x_component;
        temp.y_component = y_component - arg.y_component;
        temp.z_component = z_component - arg.z_component;
        return(temp);
       };
vec vec::operator / (double s){
        return vec(x_component/s,y_component/s,z_component/s);
       };
vec vec::operator * (double s){
        return vec(x_component*s,y_component*s,z_component*s);
       };
double vec::inner_wrt(vec other){
        return (x_component*other.x_component)\
            +(y_component*other.y_component)\
            +(z_component*other.z_component);
       };

vec vec::cross_wrt(vec other){   // a.cross_wrt(vec b) return: a cross b
        return vec(y_component*other.z_component-
z_component*other.y_component,\
            z_component*other.x_component-x_component*other.z_component, \
            x_component*other.y_component-y_component*other.x_component );
       };

double vec::distance_wrt(vec other){
        return sqrt((x_component-other.x_component)*(x_component-
other.x_component)\
            +(y_component-other.y_component)*(y_component-other.y_component)\
            +(z_component-other.z_component)*(z_component-
other.z_component));
       };

double vec::norm(){


return( sqrt(x_component*x_component+y_component*y_component+z_component
*z_component) );
```

```
};

vec vec::rotate(vec arg_axis, double arg_angle){
        vec v_final;
        v_final.x_component = cos(arg_angle)*x_component \
            - sin(arg_angle)* cross_wrt(arg_axis).x_component;
        v_final.y_component = cos(arg_angle)*y_component \
            - sin(arg_angle)* cross_wrt(arg_axis).y_component;
        v_final.z_component = cos(arg_angle)*z_component \
            - sin(arg_angle)* cross_wrt(arg_axis).z_component;
        return(v_final);
};



==========  Point.h
#ifndef POINT_H
#define POINT_H

#include "header.h"
#include "vec.h"

class Point{
        vec X, p_normal, X_A;
        double beta;
public:
        ////////////////////
        //specify values////
        ////////////////////
        Point(){beta = 0;}; //declare the constructor, which is envoked when
        //creating an instance: vec instance;


        ////////////////////////
        ///set member data/////
        ////////////////////////
        void set_X(vec other){
          X = other;
        };
        void set_XA_N_X(){
          vec Xtemp;
          vec z_hat(0,0,1);
          Xtemp = X - z_hat*(X.inner_wrt(z_hat));
```

```
        X_A = (Xtemp/Xtemp.norm())*R1; //set X_A
        Xtemp = X - X_A;
        p_normal = Xtemp/Xtemp.norm();  //set p_normal
        X = X_A + p_normal*R2;  //calibrate X, st it is ON the torus
      };
      void set_beta(double other){
        beta = other;
      };

      //////////////////////
      ///visit member data///
      //////////////////////

      vec get_X(){
        return X;
      };
      vec get_X_A(){
        return X_A;
      };
      vec get_p_normal(){
        return p_normal;
      };
      double get_beta(){
        return beta;
      };


};



#endif
```

Codes for the paper “Dynamics of vacancies in two-dimensional Lennard-Jones crystals”

======== LAMMPS codes

LAMMPS simualtions of Crystal Melting
start on April 28, 2014

variable radius equal e-6 # meter

variable imax equal 2
print "imax = ${imax}"

variable i loop ${imax}
    # i from 1 to imax, inclusive; ref: manual/variable

variable n_dump equal ${n_run}/50
    # the denominator # * (imax +1) of frames in the exported video

variable n_print equal ${n_run}
    # 10 prints

variable size_hole_x equal 10.0 ## vary it
variable size_hole_y equal 1.0 ##fix it

##################

units si # standard international: meter, joul, seconds etc

atom_style charge # atomic # # sphere # diameter, mass, angular velocity

boundary p p p
    # the simulation box should be periodic in z for 2D simulations. the thickness (along z) should be narrow, finite

dimension 2

```
newton on off # pairwise / bonded interactions



lattice hex 5.0*${radius}
        # lattice constant in distance units

variable size_box equal 100*${radius} # 100
variable position_cylinder equal ${size_box}/2
variable size_cylinder equal ${size_box}/4   # ${size_box}/100

region mybox block     0  ${size_box}  0  ${size_box}  -0.5*${radius} 0.5*${radius}


create_box 1  mybox


create_atoms 1    region mybox #  crystal_core


set type 1 charge 1.6e-16 #1000|e| per colloid


mass 1 4.0e-15  # kg




variable temp_for_velocity equal 200

velocity all create ${temp_for_velocity}  4928459 rot yes dist gaussian
        # create reduced temperature = (the first number) * epsilon,
        # random number of seeds = 4928...
        # rot yes: the angular momentum is zeroed
        # dist gaussian: to generate velocity with a mean of 0.0 and a sigma scaled
to produce the requested temperature


##############################################
##############################################
################start: set up potential
##############################################
```

```
pair_style coul/debye 0.001*${radius} 10.0*${radius}
        # first number: kappa
        # second number: cut-off

dielectric 1.0

pair_coeff 1  1



variable temp_1  equal 200  # kelvin. 300K room temp  273.15
variable temp_2  equal 350
variable n_run equal 10000


##############################################################
##############################################################
################finish: set up potential
##############################################################


timestep    1.0e-8 # default value for units si


fix 1    all enforce2d
        # Zero out the z-dimension velocity and force on each atom in the group
        # 将该 fix 作为最后一个 fix 命令的原因是将其它 fix 命令引起的力的变化均清零
        # LAMMPS 中给出的许多程序实例都是针对二维模拟的


thermo_style custom step temp epair etotal press vol
        # to print out thermodynamic data to the screen and log file
thermo      ${n_print}
        # Compute and print thermodynamic info every N time steps


dump 1     all custom ${n_dump} out.lammpstrj id type xu yu zu
        # dump every this many timesteps
```

```
fix 2    all npt temp ${temp_1} ${temp_1} 10.0  iso 0.0 0.0 100.0
         # Tstart Tstop damp
run     ${n_run}
unfix 2



print "0${temp_1}" file real_temp.txt screen no
print "0${temp_1}" file set_temp.txt screen no

################### creat a loop

label start_of_loop_1
         # Label this line of the input script with the chosen ID.
         # to play together with "jump"

print "Iteration i = $i"

variable my_temp_1 equal ${temp_1}+(${temp_2}-${temp_1})*($i-1)/(${imax}-1)
variable my_temp_2 equal ${temp_1}+(${temp_2}-${temp_1})*$i/(${imax}-1)
# print "my_temp_1 = ${my_temp_1}"
# print "my_temp_2 = ${my_temp_2}"
# print ""




fix 2    all npt temp ${my_temp_1} ${my_temp_2} 10.0  iso 0.0 0.0 100.0
# dump_modify 1 append yes
run     ${n_run}
unfix 2

variable t1 equal temp
         # Ref: http://lammps.sandia.gov/doc/variable.html
         # The previous run invokes various computes,
         # including the one for temperature, so that the value it stores
         # is current and can be accessed by the variable "t2" after the run has
completed.
print "${i} ${t1} " append real_temp.txt screen no
print "${i} ${my_temp_2} " append set_temp.txt screen no


next i
```

```
fix 2    all npt temp ${my_temp_2} ${my_temp_2} 10.0  iso 0.0 0.0 100.0
# dump_modify 1 append yes
run     ${n_run}
unfix 2

variable t2 equal temp
print "${i} ${t2} " append real_temp.txt screen no
print "${i} ${my_temp_2} " append set_temp.txt screen no



next i

jump SELF start_of_loop_1
        # lmp_g++ -in in.script, ref: manual/jump

# end of the loop




# write_restart restart.equil
```

Codes for the paper “Topological Defects in Flat Geometry: The Role of Density Inhomogeneity”

========== cpp:

```
#include "variables.h"
#include <sstream>

double Vij (vec arg1, vec arg2);
vec move_s (vec arg_r, vec arg_s);

double Vi (vec arg_vi, int arg_i);
double Vi_new (vec arg_vi, int arg_i);

void update_config(int arg_i);



std::string make_output_filename(int index) {
  std::ostringstream ss;
  ss << "output_files/" << index << "state"<<"/points_coordinates_" << index <<
".txt";
  return ss.str();
}

std::string make_output_filename_2(int index_1, int index_2) {
  std::ostringstream ss2;
  ss2 << "output_files/" << index_1 << "state"<<"/energy_fixed_R_" << index_2
<< ".txt";
  return ss2.str();
}


//////////////////////////////////////////////////////
//////////////////////////////////////////////////////
/////main function//////////////////////////////////
//////////////////////////////////////////////////////

using namespace std;

int main(){
```

```
  cout<<"N = "<<N<<"; s = "<<s<<endl;

        int i, j;
        R = 1.0;
        int i_compress, N_compress = 10;
        double energy[N_compress+1]; //lowest energy for one radius
        vec v_temp;
        double pro;

        //to initialize points from 1 to N. randomly on a disk of radius R = 1
        ofstream coordinates_ifile, energy_ifile;
        coordinates_ifile.open(make_output_filename(0).c_str());

        /* initialize random seed: */
        srand (time(NULL));

        for (i=1; i<=N; i++){
              pro = rand()%100000;  //rand()%10: return a random number between 0
and 9
        pro = pro/100000 -0.5;  //random number in [0,1] -0.5 = [-0.5,0.5]
          v_temp.set_x(pro*2*R);
          pro = rand()%100000;  //rand()%10: return a random number between 0
and 9
          pro = pro/100000 -0.5;  //random number in [0,1] -0.5 = [-0.5,0.5]
          v_temp.set_y(pro*2*sqrt(pow(R,2)-pow(v_temp.get_x(),2)));
          pt[i].set_X(v_temp);
          pt_new[i].set_X(v_temp);
          coordinates_ifile<<pt[i].get_X().get_x()<<"\t"<<pt[i].get_X().get_y()<<endl;
        }
        coordinates_ifile.close();

        /////energy of initial state////
        energy[0] = 0;
        for(i = 1; i <= N; i++){
          energy[0] = energy[0] + Vi(pt[i].get_X(), i);
        }
        //energy[0] = energy_ini;
        cout<<"the energy of initial state is: "<<energy[0]<<endl;

        double energy_t, energy_old, energy_new, energy_after_sweep,
energy_before_sweep;
        int count = 0;

        ////////////////////////////////////////////////////////////////////////
```

```
//relax from the initial random configuration (0) by force method////////
//////////////////////////////////////////////////////////////////////////


 do {

  // calculate energy before sweep
  energy_t = 0.0;
  for(j = 1; j <= N; j++){
      energy_t = energy_t  + Vi(pt[j].get_X(), j);
  }
  energy_before_sweep = energy_t;


 //a sweep
 for(i = 1; i <= N; i++){
  update_config(i); //pt[i] unchanged. pt_new[i] generated.

  // calculate old energy
  energy_t = 0.0;
  for(j = 1; j <= N; j++){
      energy_t = energy_t  + Vi(pt[j].get_X(), j);
  }
  energy_old = energy_t;

  // calculate new energy
  energy_t = 0.0;
  for(j = 1; j <= N; j++){
    energy_t = energy_t  + Vi_new(pt_new[j].get_X(), j);
  }
  energy_new = energy_t;

  if (energy_new < energy_old) {
    pt[i].set_X(pt_new[i].get_X());
    //continue;
  }
  else {
    pt_new[i].set_X(pt[i].get_X());
  }
 } // end of a sweep
  count ++;

  // calculate energy after sweep
```

```
            energy_t = 0.0;
            for(j = 1; j <= N; j++){
                energy_t = energy_t  + Vi(pt[j].get_X(), j);
            }
            energy_after_sweep = energy_t;


          //record particle position
          if (count%1 == 0){
            coordinates_ifile.open(make_output_filename(count).c_str()); // "1" is the
ground state at R = 1.0
            for(i = 1; i <= N; i++){

coordinates_ifile<<pt[i].get_X().get_x()<<"\t"<<pt[i].get_X().get_y()<<endl;
            }
            coordinates_ifile.close();
          }

          //record energy reduction
           energy_ifile.open("output_files/energy_sweeps.txt",ios::app | ios::ate);
           energy_ifile<<count<<"\t"<<energy_after_sweep<<endl;
           energy_ifile.close();


           if (count%1 == 0)
               cout<<"at sweep: "<<count<<"\t"<<energy_after_sweep<<"\t"<<\
                 energy_before_sweep<<" after - before = "<<energy_after_sweep-
energy_before_sweep<<endl;

           } while ( energy_after_sweep < energy_before_sweep ); //count < 10000
energy_new < energy_old




          ////////////////////////////////////
          ///write to files//////////////////
          ////////////////////////////////////
          ofstream parameter_ifile;

          parameter_ifile.open("output_files/parameters.txt");
          parameter_ifile<<"step length: s = "<<s<<" ; R = "<<R<<" ; N =
"<<N<<endl;
          parameter_ifile.close();
```

```
   return 0;
}

============= Point.h
#ifndef POINT_H
#define POINT_H

#include "variables.h"

class Point{
        vec X, s_vec;  //position of a point is X and its associated s_vec generate a
new point: X+s_vec.
public:
        ////////////////////
        //specify values////
        ////////////////////


        ////////////////////////
        ///set member data/////
        ////////////////////////
        void set_X(vec other){
          X = other;
        };
        void set_s_vec(vec other){
          s_vec = other;
        };


        ////////////////////////
        ///visit member data///
        ////////////////////////

        vec get_X(){
          return X;
        };
        vec get_s_vec(){
          return s_vec;
        };

};
```

```
#endif
```

========== Vij.cpp

```
#include "variables.h"


double Vij (vec arg1, vec arg2){
        double arg_d;
        vec diff;
        diff =  arg1 - arg2;
        arg_d = 1.0/diff.norm(); //-log(diff.norm());
        return arg_d;
};
```

==========vec.h

```
#include "vec.h"


double vec::get_x(){
         return x_component;
        };
double vec::get_y(){
         return y_component;
        };


vec vec::operator + (vec arg){
         vec temp;
         temp.x_component = x_component + arg.x_component;
         temp.y_component = y_component + arg.y_component;
         return(temp);
        };
vec vec::operator - (vec arg){
         vec temp;
         temp.x_component = x_component - arg.x_component;
         temp.y_component = y_component - arg.y_component;
         return(temp);
        };
vec vec::operator / (double arg){
         vec v_arg;
```

```
        v_arg.x_component = x_component/arg;
        v_arg.y_component = y_component/arg;
        return v_arg;
       };
vec vec::operator * (double arg){
        vec v_arg;
        v_arg.x_component = x_component*arg;
        v_arg.y_component = y_component*arg;
        return v_arg;
       };
double vec::inner_wrt(vec other){
        return (x_component*other.x_component)\
             +(y_component*other.y_component);
};


double vec::distance_wrt(vec other){
        return sqrt((x_component-other.x_component)*(x_component-
other.x_component)\
             +(y_component-other.y_component)*(y_component-
other.y_component));
       };

double vec::norm(){
        return( sqrt(x_component*x_component+y_component*y_component) );
};

==========variables.h

#ifndef _HEADER_H
#define _HEADER_H
#include <iostream>
#include <string>
//#include <vector>
#include <cmath>
#include <cstdlib>
#include "vec.h"
#include "Point.h"
#include <time.h>
#include <fstream>

using namespace std;

// Constants & Input
```

```
#define Pi    (3.1415926536)  // PI value

#define s     (0.01)     // step length
//#define R     (1.0)          // radius of disk
#define N       (600)           //total number of particles on disk

extern Point pt[N+1];
extern Point pt_new[N+1];
//extern double energy_ini, denergy;
extern double R;

#endif
```

==========

1.

1. Report Type

Final Report

Primary Contact E-mail
Contact email if there is a problem with the report.

m-olvera@northwestern.edu

Primary Contact Phone Number
Contact phone number if there is a problem with the report

8474917801

Organization / Institution name

Northwestern University

Grant/Contract Title
The full title of the funded effort.

Paradigms for Emergence of Shape and Function in Biomolecular Electrolytes for the Design of Biomimetic Materials.

Grant/Contract Number
AFOSR assigned control number. It must begin with "FA9550" or "F49620" or "FA2386".

FA9550-10-1-0167

Principal Investigator Name
The full name of the principal investigator on the grant or contract.

Monica Olvera de la Cruz

Program Manager
The AFOSR Program Manager currently assigned to the award

Dr. Julie Moses

Reporting Period Start Date

05/01/2010

Reporting Period End Date

08/31/2015

Abstract

We discovered the buckling of elastically heterogeneous nano-containers into regular and irregular polyhedral geometries observed in organelles of multicomponent proteins and in halophilic organisms. We determined the physical properties of ionic membranes. By co-assembling cationic and anionic amphiphiles programmed to form ionic crystalline vesicles we demonstrated the buckling model and produced biomimetic organelles. We extended the model to describe the blebbing of nuclear lamina in cells in patients with cancer and other serious pathologies, and discovered that blebbing is due to the segregation of two main lamins types that form the elastic membranes that protect the nucleus of cells. We discovered long-range attractions mediated by concentrated electrolytes. These depletion-like interactions are responsible for the well known “salting out” phenomena. We provided the rules for efficient self-replication of colloidal dimers. We demonstrated that the input of cyclic energy leads to maximum exponential growth. We discovered the functional required to performed molecular dynamics studies of charged components in dielectrically heterogeneous media, and developed efficient codes to analyze concentrated electrolyte solutions. We trained a large number of students and postdocs in soft-matter physics. We bought a computer cluster that allowed us to carry out these studies.

Distribution Statement

This is block 12 on the SF298 form.

Distribution A - Approved for Public Release

Explanation for Distribution Statement

If this is not approved for public release, please provide a short explanation. E.g., contains proprietary information.

SF298 Form

Please attach your SF298 form. A blank SF298 can be found here. Please do not password protect or secure the PDF The maximum file size for an SF298 is 50MB.

sf298.pdf

Upload the Report Document. File must be a PDF. Please do not password protect or secure the PDF . The maximum file size for the Report Document is 50MB.

Olvera_FinalReport.pdf

Upload a Report Document, if any. The maximum file size for the Report Document is 50MB.

Archival Publications (published) during reporting period:

Guerrero-Garcia GI, Gonzalez-Mozuelos P, de la Cruz MO. Potential of mean force between identical charged nanoparticles immersed in a size-asymmetric monovalent electrolyte. Journal of Chemical Physics. 2011;135(16).

2. Guo PJ, Sknepnek R, de la Cruz MO. Electrostatic-Driven Ridge Formation on Nanoparticles Coated with Charged End-Group Ligands. Journal of Physical Chemistry C. 2011;115(14):6484-90.

3. Jha PK, Zwanikken JW, Detcheverry FA, de Pablo JJ, de la Cruz MO. Study of volume phase transitions in polymeric nanogels by theoretically informed coarse-grained simulations. Soft Matter. 2011;7(13):5965-75.

4. Kuzovkov VN, Kotomin EA, de la Cruz MO. The non-equilibrium charge screening effects in diffusion-driven systems with pattern formation. Journal of Chemical Physics. 2011;135(3).

5. Vernizzi G, Sknepnek R, de la Cruz MO. Platonic and Archimedean geometries in multicomponent elastic membranes. Proceedings of the National Academy of Sciences of the United States of America. 2011;108(11):4292-6.

6. Vernizzi G, Zhang DS, de la Cruz MO. Structural phase transitions and mechanical properties of binary ionic colloidal crystals at interfaces. Soft Matter. 2011;7(13):6285-93.

7. Zwanikken JW, Guo PJ, Mirkin CA, de la Cruz MO. Local Ionic Environment around Polyvalent Nucleic Acid-Functionalized Nanoparticles. Journal of Physical Chemistry C. 2011;115(33):16368-73.

8. Zwanikken JW, Jha PK, de la Cruz MO. A practical integral equation for the structure and thermodynamics of hard sphere Coulomb fluids. Journal of Chemical Physics. 2011;135(6).

9. Demers MF, Sknepnek R, de la Cruz MO. Curvature-driven effective attraction in multicomponent membranes. Physical Review E. 2012;86(2).

10. Jadhao V, Solis FJ, de la Cruz MO. Simulation of Charged Systems in Heterogeneous Dielectric Media via a True Energy Functional. Physical Review Letters. 2012;109(22).

11. Leung CY, Palmer LC, Qiao BF, Kewalramani S, Sknepnek R, Newcomb CJ, et al. Molecular Crystallization Controlled by pH Regulates Mesoscopic Membrane Morphology. Acs Nano. 2012;6(12):10901-9.

12. Li T, Sknepnek R, Macfarlane RJ, Mirkin CA, de la Cruz MO. Modeling the Crystallization of Spherical Nucleic Acid Nanoparticle Conjugates with Molecular Dynamics Simulations. Nano Letters. 2012;12(5):2509-14.

13. Sknepnek R, Vernizzi G, de la Cruz MO. Charge renormalization of bilayer elastic properties. Journal of Chemical Physics. 2012;137(10).

14. Sknepnek R, Vernizzi G, de la Cruz MO. Buckling of multicomponent elastic shells with line tension. Soft Matter. 2012;8(3):636-44.

15. Su JY, de la Cruz MO, Guo HX. Solubility and transport of cationic and anionic patterned nanoparticles. Physical Review E. 2012;85(1).

16. Yao ZW, Sknepnek R, Thomas CK, de la Cruz MO. Shapes of pored membranes. Soft Matter. 2012;8(46):11613-9.

17. Ziebert F, Swaminathan S, Aranson IS. Model for self-polarization and motility of keratocyte fragments. Journal of the Royal Society Interface. 2012;9(70):1084-92.
18. Funkhouser CM, Sknepnek R, de la Cruz MO. Topological defects in the buckling of elastic membranes. Soft Matter. 2013;9(1):60-8.
19. Funkhouser CM, Sknepnek R, Shimi T, Goldman AE, Goldman RD, de la Cruz MO. Mechanical model of blebbing in nuclear lamin meshworks. Proceedings of the National Academy of Sciences of the United States of America. 2013;110(9):3248-53.
20. Gonzalez-Mozuelos P, Guerrero-Garcia GI, de la Cruz MO. An exact method to obtain effective electrostatic interactions from computer simulations: The case of effective charge amplification. Journal of Chemical Physics. 2013;139(6).
21. Guerrero-Garcia GI, de la Cruz MO. Inversion of the Electric Field at the Electrified Liquid-Liquid Interface. Journal of Chemical Theory and Computation. 2013;9(1):1-7.
22. Guerrero-Garcia GI, Gonzalez-Mozuelos P, de la Cruz MO. Large Counterions Boost the Solubility and Renormalized Charge of Suspended Nanoparticles. Acs Nano. 2013;7(11):9714-23.
23. Guerrero-Garcia GI, Jing YF, de la Cruz MO. Enhancing and reversing the electric field at the oil-water interface with size-asymmetric monovalent ions. Soft Matter. 2013;9(26):6046-52.
24. Jadhao V, Solis FJ, de la Cruz MO. Free-energy functionals of the electrostatic potential for Poisson-Boltzmann theory. Physical Review E. 2013;88(2).
25. Jadhao V, Solis FJ, de la Cruz MO. A variational formulation of electrostatics in a medium with spatially varying dielectric permittivity. Journal of Chemical Physics. 2013;138(5).
26. Kewalramani S, Zwanikken JW, Macfarlane RJ, Leung CY, de la Cruz MO, Mirkin CA, et al. Counterion Distribution Surrounding Spherical Nucleic Acid-Au Nanoparticle Conjugates Probed by Small-Angle X-ray Scattering. Acs Nano. 2013;7(12):11301-9.
27. Leung CY, Palmer LC, Kewalramani S, Qiao BF, Stupp SI, de la Cruz MO, et al. Crystalline polymorphism induced by charge regulation in ionic membranes. Proceedings of the National Academy of Sciences of the United States of America. 2013;110(41):16309-14.
28. Li T, Sknepnek R, de la Cruz MO. Thermally Active Hybridization Drives the Crystallization of DNA-Functionalized Nanoparticles. Journal of the American Chemical Society. 2013;135(23):8535-41.
29. Parsaeian A, de la Cruz MO, Marko JF. Binding-rebinding dynamics of proteins interacting nonspecifically with a long DNA molecule. Physical Review E. 2013;88(4).
30. Qiao BF, de la Cruz MO. Driving Force for Water Permeation Across Lipid Membranes. Journal of Physical Chemistry Letters. 2013;4(19):3233-7.
31. Qiao BF, de la Cruz MO. Driving Force for Crystallization of Anionic Lipid Membranes Revealed by Atomistic Simulations. Journal of Physical Chemistry B. 2013;117(17):5073-80.
32. Sing CE, Zwanikken JW, de la Cruz MO. Interfacial Behavior in Polyelectrolyte Blends: Hybrid Liquid-State Integral Equation and Self-Consistent Field Theory Study. Physical Review Letters. 2013;111(16).
33. Sing CE, Zwanikken JW, de la Cruz MO. Effect of Ion-Ion Correlations on Polyelectrolyte Gel Collapse and Reentrant Swelling. Macromolecules. 2013;46(12):5053-65.
34. Sing CE, Zwanilden JW, de la Cruz MO. Ion Correlation-Induced Phase Separation in Polyelectrolyte Blends. Acs Macro Letters. 2013;2(11):1042-6.
35. Solis FJ, Jadhao V, de la Cruz MO. Generating true minima in constrained variational formulations via modified Lagrange multipliers. Physical Review E. 2013;88(5).
36. Thomas CK, de la Cruz MO. Theory and simulations of crystalline control via salinity and pH in ionizable membranes. Soft Matter. 2013;9(2):429-34.
37. Yao ZW, de la Cruz MO. Topological Defects in Flat Geometry: The Role of Density Inhomogeneity. Physical Review Letters. 2013;111(11).
38. Yao ZW, de la Cruz MO. Electrostatic repulsion-driven crystallization model arising from filament networks. Physical Review E. 2013;87(4).
39. Yao ZW, de la Cruz MO. Packing of charged chains on toroidal geometries. Physical Review E. 2013;87(1).
40. Yao ZW, Qiao BF, de la Cruz MO. Potassium ions in the cavity of a KcsA channel model. Physical Review E. 2013;88(6).
41. Zhang R, Jha PK, de la Cruz MO. Non-equilibrium ionic assemblies of oppositely charged

nanoparticles. Soft Matter. 2013;9(20):5042-51.
42. Zwanikken JW, de la Cruz MO. Tunable soft structure in charged fluids confined by dielectric interfaces. Proceedings of the National Academy of Sciences of the United States of America. 2013;110(14):5301-8.
43. Auyeung E, Li T, Senesi AJ, Schmucker AL, Pals BC, de la Cruz MO, et al. DNA-mediated nanoparticle crystallization into Wulff polyhedra. Nature. 2014;505(7481):73-7.
44. Garcia GIG, de la Cruz MO. Polarization Effects of Dielectric Nanoparticles in Aqueous Charge-Asymmetric Electrolytes. Journal of Physical Chemistry B. 2014;118(29):8854-62.
45. Jadhao V, Thomas CK, de la Cruz MO. Electrostatics-driven shape transitions in soft shells. Proceedings of the National Academy of Sciences of the United States of America. 2014;111(35):12673-8.
46. Ovanesyan Z, Medasani B, Fenley MO, Guerrero-Garcia GI, de la Cruz MO, Marucho M. Excluded volume and ion-ion correlation effects on the ionic atmosphere around B-DNA: Theory, simulations, and experiments. Journal of Chemical Physics. 2014;141(22).
47. Qiao B, Demars T, de la Cruz MO, Ellis RJ. How Hydrogen Bonds Affect the Growth of Reverse Micelles around Coordinating Metal Ions. Journal of Physical Chemistry Letters. 2014;5(8):1440-4.
48. Sing CE, de la Cruz MO. Polyelectrolyte Blends and Nontrivial Behavior in Effective Flory-Huggins Parameters. Acs Macro Letters. 2014;3(8):698-702.
49. Sing CE, de la Cruz MO, Marko JF. Multiple-binding-site mechanism explains concentration-dependent unbinding rates of DNA-binding proteins. Nucleic Acids Research. 2014;42(6):3783-91.
50. Yao ZW, de la Cruz MO. Dynamics of vacancies in two-dimensional Lennard-Jones crystals. Physical Review E. 2014;90(6).
51. Yao ZW, de La Cruz MO. Polydispersity-driven topological defects as order-restoring excitations. Proceedings of the National Academy of Sciences of the United States of America. 2014;111(14):5094-9.
52. Zhang R, Dempster JM, de la Cruz MO. Self-replication in colloids with asymmetric interactions. Soft Matter. 2014;10(9):1315-9.
53. Boon N, Guerrero-Garcia GI, van Roij R, de la Cruz MO. Effective charges and virial pressure of concentrated macroion solutions. Proceedings of the National Academy of Sciences of the United States of America. 2015;112(30):9242-6.
54. Dempster JM, Zhang R, de la Cruz MO. Self-replication with magnetic dipolar colloids. Physical Review E. 2015;92(4).
55. Jadhao V, Yao ZW, Thomas CK, de la Cruz MO. Coulomb energy of uniformly charged spheroidal shell systems. Physical Review E. 2015;91(3).
56. Kwon HK, Zwanikken JW, Shull KR, de la Cruz MO. Theoretical Analysis of Multiple Phase Coexistence in Polyelectrolyte Blends. Macromolecules. 2015;48(16):6008-15.

Changes in research objectives (if any):

Change in AFOSR Program Manager, if any:

Extensions granted or milestones slipped, if any:

AFOSR LRIR Number

LRIR Title

Reporting Period

Laboratory Task Manager

Program Officer

Research Objectives

Technical Summary

Funding Summary by Cost Category (by FY, $K)

	Starting FY	FY+1	FY+2
Salary			
Equipment/Facilities			
Supplies			
Total			

Report Document

Report Document - Text Analysis

Report Document - Text Analysis

Appendix Documents

2. Thank You

E-mail user

Nov 30, 2015 23:02:40 Success: Email Sent to: m-olvera@northwestern.edu